triSYCL implementation of OpenCL SYCL

Generated by Doxygen 1.8.13

Contents

1 Main Page **1**

2 Todo List **3**

3 Module Index **15**

3.1 Modules 15

4 Namespace Index **17**

4.1 Namespace List 17

5 Hierarchical Index **19**

5.1 Class Hierarchy 19

6 Class Index **25**

6.1 Class List 25

7 File Index **27**

7.1 File List 27

8 Module Documentation **29**

8.1 Data access and storage in SYCL 29

8.1.1 Detailed Description 31

8.1.2 Class Documentation 31

8.1.2.1 class cl::sycl::detail::accessor_base 31

8.1.2.2 class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local > . . 34

8.1.2.3 class cl::sycl::accessor 50

8.1.2.4 class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe > . . . 66

8.1.2.5 class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe > . . . 69
8.1.2.6 class cl::sycl::detail::accessor . . . 72
8.1.2.7 class cl::sycl::detail::buffer . . . 94
8.1.2.8 struct cl::sycl::detail::buffer_base . . . 111
8.1.2.9 class cl::sycl::detail::buffer_waiter . . . 123
8.1.2.10 class cl::sycl::buffer . . . 125
8.1.2.11 struct cl::sycl::image . . . 143
8.1.2.12 struct cl::sycl::detail::reserve_id . . . 144
8.1.2.13 class cl::sycl::detail::pipe . . . 146
8.1.2.14 class cl::sycl::detail::pipe_accessor . . . 160
8.1.2.15 class cl::sycl::pipe . . . 171
8.1.2.16 class cl::sycl::detail::pipe_reservation . . . 174
8.1.2.17 struct cl::sycl::pipe_reservation . . . 185
8.1.2.18 class cl::sycl::static_pipe . . . 197
8.1.3 Typedef Documentation . . . 201
8.1.3.1 buffer_allocator . . . 201
8.1.3.2 image_allocator . . . 202
8.1.3.3 map_allocator . . . 202
8.1.4 Function Documentation . . . 202
8.1.4.1 add_buffer_to_task() . . . 202
8.1.4.2 buffer_add_to_task() . . . 203
8.1.4.3 get_pipe_detail() . . . 203
8.1.4.4 waiter() . . . 204
8.2 Dealing with OpenCL address spaces . . . 205
8.2.1 Detailed Description . . . 207
8.2.2 Class Documentation . . . 207
8.2.2.1 struct cl::sycl::detail::ocl_type . . . 207
8.2.2.2 struct cl::sycl::detail::ocl_type< T, constant_address_space > . . . 207
8.2.2.3 struct cl::sycl::detail::ocl_type< T, global_address_space > . . . 208
8.2.2.4 struct cl::sycl::detail::ocl_type< T, local_address_space > . . . 208

8.2.2.5 struct cl::sycl::detail::ocl_type< T, private_address_space > . . . 209
8.2.2.6 struct cl::sycl::detail::address_space_array . . . 209
8.2.2.7 struct cl::sycl::detail::address_space_fundamental . . . 212
8.2.2.8 struct cl::sycl::detail::address_space_object . . . 215
8.2.2.9 struct cl::sycl::detail::address_space_ptr . . . 218
8.2.2.10 struct cl::sycl::detail::address_space_base . . . 221
8.2.2.11 struct cl::sycl::detail::address_space_variable . . . 223
8.2.3 Typedef Documentation . . . 227
8.2.3.1 addr_space . . . 227
8.2.3.2 constant . . . 227
8.2.3.3 constant_ptr . . . 228
8.2.3.4 generic . . . 228
8.2.3.5 global . . . 228
8.2.3.6 global_ptr . . . 229
8.2.3.7 local . . . 229
8.2.3.8 local_ptr . . . 229
8.2.3.9 multi_ptr . . . 230
8.2.3.10 priv . . . 230
8.2.3.11 private_ptr . . . 230
8.2.4 Enumeration Type Documentation . . . 231
8.2.4.1 address_space . . . 231
8.2.5 Function Documentation . . . 231
8.2.5.1 make_multi() . . . 231
8.3 Platforms, contexts, devices and queues . . . 233
8.3.1 Detailed Description . . . 236
8.3.2 Class Documentation . . . 236
8.3.2.1 class cl::sycl::detail::context . . . 236
8.3.2.2 class cl::sycl::detail::host_context . . . 239
8.3.2.3 class cl::sycl::context . . . 243
8.3.2.4 class cl::sycl::detail::device . . . 253

8.3.2.5 class cl::sycl::device . . . 256
8.3.2.6 class cl::sycl::device_type_selector . . . 267
8.3.2.7 class cl::sycl::device_typename_selector . . . 270
8.3.2.8 class cl::sycl::device_selector . . . 272
8.3.2.9 class cl::sycl::handler . . . 274
8.3.2.10 class cl::sycl::detail::kernel . . . 284
8.3.2.11 class cl::sycl::kernel . . . 287
8.3.2.12 class cl::sycl::detail::host_platform . . . 291
8.3.2.13 class cl::sycl::detail::opencl_platform . . . 295
8.3.2.14 class cl::sycl::detail::platform . . . 300
8.3.2.15 class cl::sycl::platform . . . 302
8.3.2.16 class cl::sycl::queue . . . 310
8.3.3 Typedef Documentation . . . 321
8.3.3.1 cpu_selector . . . 322
8.3.3.2 default_selector . . . 322
8.3.3.3 device_exec_capabilities . . . 322
8.3.3.4 device_fp_config . . . 323
8.3.3.5 device_queue_properties . . . 323
8.3.3.6 gpu_selector . . . 323
8.3.3.7 host_selector . . . 323
8.3.4 Enumeration Type Documentation . . . 323
8.3.4.1 context . . . 323
8.3.4.2 device . . . 324
8.3.4.3 device_affinity_domain . . . 327
8.3.4.4 device_execution_capabilities . . . 327
8.3.4.5 device_partition_property . . . 328
8.3.4.6 device_partition_type . . . 328
8.3.4.7 device_type . . . 329
8.3.4.8 fp_config . . . 329
8.3.4.9 global_mem_cache_type . . . 330

8.3.4.10 local_mem_type . . . 330
8.3.4.11 platform . . . 331
8.3.5 Function Documentation . . . 332
8.3.5.1 device::get_info< info::device::device_type >() . . . 332
8.3.5.2 device::get_info< info::device::local_mem_size >() . . . 332
8.3.5.3 device::get_info< info::device::max_compute_units >() . . . 333
8.3.5.4 device::get_info< info::device::max_mem_alloc_size >() . . . 333
8.3.5.5 device::get_info< info::device::max_work_group_size >() . . . 333
8.3.5.6 device::get_info< info::device::name >() . . . 334
8.3.5.7 device::get_info< info::device::profile >() . . . 334
8.3.5.8 device::get_info< info::device::vendor >() . . . 334
8.3.5.9 get_devices() [1/3] . . . 334
8.3.5.10 get_devices() [2/3] . . . 336
8.3.5.11 get_devices() [3/3] . . . 337
8.3.6 Variable Documentation . . . 337
8.3.6.1 TRISYCL_WEAK_ATTRIB_SUFFIX . . . 337
8.4 Helpers to do array and tuple conversion . . . 338
8.4.1 Detailed Description . . . 338
8.4.2 Class Documentation . . . 338
8.4.2.1 struct cl::sycl::detail::expand_to_vector . . . 338
8.4.2.2 struct cl::sycl::detail::expand_to_vector< V, Tuple, true > . . . 339
8.4.3 Function Documentation . . . 339
8.4.3.1 expand() [1/3] . . . 339
8.4.3.2 expand() [2/3] . . . 340
8.4.3.3 expand() [3/3] . . . 340
8.4.3.4 fill_tuple() . . . 341
8.4.3.5 tuple_to_array() . . . 341
8.4.3.6 tuple_to_array_iterate() . . . 342
8.5 Some helpers for the implementation . . . 343
8.5.1 Detailed Description . . . 343

8.5.2 Class Documentation . . . 343
8.5.2.1 struct cl::sycl::detail::container_element_aspect . . . 343
8.5.2.2 struct cl::sycl::detail::small_array . . . 344
8.5.2.3 struct cl::sycl::detail::small_array_123 . . . 352
8.5.2.4 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 > . . . 353
8.5.2.5 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 > . . . 355
8.5.2.6 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 > . . . 358
8.5.3 Macro Definition Documentation . . . 360
8.5.3.1 TRISYCL_BOOST_OPERATOR_VECTOR_OP . . . 360
8.5.3.2 TRISYCL_LOGICAL_OPERATOR_VECTOR_OP . . . 361
8.5.4 Function Documentation . . . 361
8.5.4.1 linear_id() . . . 361
8.5.4.2 unimplemented() . . . 362
8.6 Debugging and tracing support . . . 364
8.6.1 Detailed Description . . . 364
8.6.2 Class Documentation . . . 364
8.6.2.1 struct cl::sycl::detail::debug . . . 364
8.6.2.2 struct cl::sycl::detail::display_vector . . . 364
8.6.3 Function Documentation . . . 365
8.6.3.1 trace_kernel() . . . 366
8.7 Manage default configuration and types . . . 367
8.7.1 Detailed Description . . . 367
8.7.2 Macro Definition Documentation . . . 367
8.7.2.1 __SYCL_SINGLE_SOURCE__ . . . 367
8.7.2.2 CL_SYCL_LANGUAGE_VERSION . . . 368
8.7.2.3 SDX_KERNEL_PROGRAM_OWNING_BUG . . . 368
8.7.2.4 TRISYCL_CL_LANGUAGE_VERSION . . . 368
8.7.2.5 TRISYCL_DEVICE_ONLY . . . 368
8.7.2.6 TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE . . . 368
8.7.2.7 TRISYCL_SKIP_OPENCL . . . 368

8.8 Device-side runtime implementation . . . 369
8.8.1 Detailed Description . . . 369
8.8.2 Class Documentation . . . 369
8.8.2.1 class cl::sycl::drt::accessor . . . 369
8.8.3 Function Documentation . . . 372
8.8.3.1 serialize_accessor_arg() . . . 372
8.8.3.2 serialize_arg() . . . 373
8.8.3.3 set_kernel() . . . 374
8.8.4 Variable Documentation . . . 375
8.8.4.1 kernel_IR . . . 375
8.9 Error handling . . . 376
8.9.1 Detailed Description . . . 377
8.9.2 Class Documentation . . . 377
8.9.2.1 struct cl::sycl::error_handler . . . 377
8.9.2.2 struct cl::sycl::exception_list . . . 379
8.9.2.3 class cl::sycl::exception . . . 379
8.9.2.4 class cl::sycl::cl_exception . . . 382
8.9.2.5 struct cl::sycl::async_exception . . . 384
8.9.2.6 class cl::sycl::runtime_error . . . 385
8.9.2.7 class cl::sycl::kernel_error . . . 386
8.9.2.8 class cl::sycl::accessor_error . . . 387
8.9.2.9 class cl::sycl::nd_range_error . . . 388
8.9.2.10 class cl::sycl::event_error . . . 389
8.9.2.11 class cl::sycl::invalid_parameter_error . . . 390
8.9.2.12 class cl::sycl::device_error . . . 391
8.9.2.13 class cl::sycl::compile_program_error . . . 392
8.9.2.14 class cl::sycl::link_program_error . . . 394
8.9.2.15 class cl::sycl::invalid_object_error . . . 395
8.9.2.16 class cl::sycl::memory_allocation_error . . . 396
8.9.2.17 class cl::sycl::pipe_error . . . 397

8.9.2.18 class cl::sycl::platform_error . . . 398
8.9.2.19 class cl::sycl::profiling_error . . . 399
8.9.2.20 class cl::sycl::feature_not_supported . . . 400
8.9.2.21 class cl::sycl::non_cl_error . . . 401
8.9.3 Typedef Documentation . . . 402
8.9.3.1 async_handler . . . 402
8.9.3.2 exception_ptr . . . 402
8.10 Expressing parallelism through kernels . . . 403
8.10.1 Detailed Description . . . 404
8.10.2 Class Documentation . . . 404
8.10.2.1 struct cl::sycl::group . . . 404
8.10.2.2 class cl::sycl::id . . . 415
8.10.2.3 class cl::sycl::item . . . 417
8.10.2.4 struct cl::sycl::nd_item . . . 425
8.10.2.5 struct cl::sycl::nd_range . . . 443
8.10.2.6 struct cl::sycl::detail::parallel_for_iterate . . . 447
8.10.2.7 struct cl::sycl::detail::parallel_OpenMP_for_iterate . . . 448
8.10.2.8 struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id > . . . 449
8.10.2.9 class cl::sycl::range . . . 450
8.10.3 Function Documentation . . . 452
8.10.3.1 make_id() [1/4] . . . 452
8.10.3.2 make_id() [2/4] . . . 452
8.10.3.3 make_id() [3/4] . . . 453
8.10.3.4 make_id() [4/4] . . . 453
8.10.3.5 make_range() [1/4] . . . 453
8.10.3.6 make_range() [2/4] . . . 454
8.10.3.7 make_range() [3/4] . . . 454
8.10.3.8 make_range() [4/4] . . . 454
8.10.3.9 parallel_for() [1/4] . . . 455
8.10.3.10 parallel_for() [2/4] . . . 456

8.10.3.11 parallel_for() [3/4] . . . 456

8.10.3.12 parallel_for() [4/4] . . . 457

8.10.3.13 parallel_for_global_offset() . . . 458

8.10.3.14 parallel_for_work_item() . . . 459

8.10.3.15 parallel_for_workgroup() . . . 460

8.10.3.16 parallel_for_workitem() . . . 461

8.11 Vector types in SYCL . . . 464

8.11.1 Detailed Description . . . 464

8.11.2 Class Documentation . . . 464

8.11.2.1 class cl::sycl::vec . . . 464

8.11.3 Macro Definition Documentation . . . 468

8.11.3.1 TRISYCL_DEFINE_VEC_TYPE . . . 469

8.11.3.2 TRISYCL_DEFINE_VEC_TYPE_SIZE . . . 469

8.12 Xilinx vendor extensions . . . 470

8.12.1 Detailed Description . . . 470

8.12.2 Class Documentation . . . 470

8.12.2.1 struct cl::sycl::xilinx::partition_array . . . 470

8.12.3 Macro Definition Documentation . . . 477

8.12.3.1 _ssdm_op_SpecDataflowPipeline . . . 477

8.12.3.2 _ssdm_op_SpecPipeline . . . 477

8.12.3.3 _ssdm_SpecArrayPartition . . . 477

9 Namespace Documentation **479**

9.1 cl Namespace Reference . . . 479

9.1.1 Detailed Description . . . 479

9.2 cl::sycl Namespace Reference . . . 480

9.2.1 Typedef Documentation . . . 484

9.2.1.1 exception_ptr_class . . . 484

9.2.1.2 function_class . . . 485

9.2.1.3 hash_class . . . 485

9.2.1.4 mutex_class . . . 485

9.2.1.5 shared_ptr_class . . . 485

9.2.1.6 string_class . . . 485

9.2.1.7 unique_ptr_class . . . 485

9.2.1.8 vector_class . . . 486

9.2.1.9 weak_ptr_class . . . 486

9.2.2 Function Documentation . . . 486

9.2.2.1 min() . . . 486

9.2.2.2 TRISYCL_MATH_WRAP() . . . 486

9.2.2.3 TRISYCL_MATH_WRAP2s() . . . 487

9.2.3 Variable Documentation . . . 487

9.2.3.1 y . . . 487

9.2.3.2 z . . . 487

9.3 cl::sycl::access Namespace Reference . . . 488

9.3.1 Detailed Description . . . 488

9.3.2 Enumeration Type Documentation . . . 488

9.3.2.1 fence_space . . . 488

9.3.2.2 mode . . . 489

9.3.2.3 target . . . 489

9.4 cl::sycl::detail Namespace Reference . . . 490

9.4.1 Function Documentation . . . 493

9.4.1.1 __attribute__() . . . 494

9.4.1.2 add_buffer_to_task() . . . 494
9.4.1.3 set_kernel_task_marker() . . . 495
9.4.2 Variable Documentation . . . 495
9.4.2.1 prevent_arguments_from_optimization . . . 495
9.5 cl::sycl::drt Namespace Reference . . . 496
9.5.1 Detailed Description . . . 496
9.6 cl::sycl::drt::code Namespace Reference . . . 496
9.6.1 Detailed Description . . . 497
9.7 cl::sycl::info Namespace Reference . . . 497
9.7.1 Typedef Documentation . . . 498
9.7.1.1 queue_profiling . . . 498
9.7.2 Enumeration Type Documentation . . . 498
9.7.2.1 queue . . . 499
9.8 cl::sycl::trisycl Namespace Reference . . . 499
9.8.1 Detailed Description . . . 499
9.9 cl::sycl::xilinx Namespace Reference . . . 499
9.9.1 Function Documentation . . . 500
9.9.1.1 dataflow() . . . 500
9.9.1.2 pipeline() . . . 501
9.10 cl::sycl::xilinx::partition Namespace Reference . . . 501
9.10.1 Detailed Description . . . 501
9.10.2 Enumeration Type Documentation . . . 502
9.10.2.1 type . . . 502
9.11 std Namespace Reference . . . 502

10 Class Documentation 503

10.1 cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim > Struct Template Reference 503

10.1.1 Detailed Description 504

10.1.2 Member Data Documentation 504

10.1.2.1 ele_in_each_physical_mem 504

10.1.2.2 partition_dim 504

10.1.2.3 partition_type 505

10.2 cl::sycl::detail::cache< Key, Value > Class Template Reference 505

10.2.1 Detailed Description 506

10.2.2 Member Typedef Documentation 506

10.2.2.1 key_type 506

10.2.2.2 value_type 506

10.2.3 Member Function Documentation 506

10.2.3.1 get_or_register() 506

10.2.3.2 remove() 508

10.2.4 Member Data Documentation 509

10.2.4.1 c 509

10.2.4.2 m 510

10.3 cl::sycl::xilinx::partition::complete< PDim > Struct Template Reference 510

10.3.1 Detailed Description 510

10.3.2 Member Data Documentation 511

10.3.2.1 partition_dim 511

10.3.2.2 partition_type 511

10.4 cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim > Struct Template Reference 512

10.4.1 Detailed Description 512

10.4.2 Member Data Documentation 513

10.4.2.1 partition_dim 513

10.4.2.2 partition_type 513

10.4.2.3 physical_mem_num 513

10.5 cl::sycl::trisycl::default_error_handler Struct Reference 514

10.5.1 Detailed Description 514
10.5.2 Member Function Documentation 515
10.5.2.1 report_error() 515
10.6 cl::sycl::event Class Reference 515
10.6.1 Detailed Description 515
10.6.2 Constructor & Destructor Documentation 515
10.6.2.1 event() 516
10.7 handler_event Class Reference 516
10.7.1 Detailed Description 516
10.8 std::hash< cl::sycl::buffer< T, Dimensions, Allocator > > Struct Template Reference 516
10.8.1 Detailed Description 516
10.8.2 Member Function Documentation 517
10.8.2.1 operator()() 517
10.9 std::hash< cl::sycl::context > Struct Template Reference 517
10.9.1 Detailed Description 518
10.9.2 Member Function Documentation 518
10.9.2.1 operator()() 518
10.10 std::hash< cl::sycl::device > Struct Template Reference 518
10.10.1 Detailed Description 519
10.10.2 Member Function Documentation 519
10.10.2.1 operator()() 519
10.11 std::hash< cl::sycl::kernel > Struct Template Reference 519
10.11.1 Detailed Description 520
10.11.2 Member Function Documentation 520
10.11.2.1 operator()() 520
10.12 std::hash< cl::sycl::platform > Struct Template Reference 520
10.12.1 Detailed Description 521
10.12.2 Member Function Documentation 521
10.12.2.1 operator()() 521
10.13 std::hash< cl::sycl::queue > Struct Template Reference 521

10.13.1 Detailed Description . . . 522
10.13.2 Member Function Documentation . . . 522
10.13.2.1 operator()() . . . 522
10.14cl::sycl::detail::host_device Class Reference . . . 523
10.14.1 Detailed Description . . . 524
10.14.2 Member Function Documentation . . . 524
10.14.2.1 get() . . . 524
10.14.2.2 get_boost_compute() . . . 525
10.14.2.3 get_platform() . . . 525
10.14.2.4 has_extension() . . . 526
10.14.2.5 is_accelerator() . . . 526
10.14.2.6 is_cpu() . . . 527
10.14.2.7 is_gpu() . . . 527
10.14.2.8 is_host() . . . 527
10.15cl::sycl::detail::host_queue Class Reference . . . 528
10.15.1 Detailed Description . . . 529
10.15.2 Member Function Documentation . . . 529
10.15.2.1 get() . . . 529
10.15.2.2 get_boost_compute() . . . 530
10.15.2.3 get_context() . . . 530
10.15.2.4 get_device() . . . 530
10.15.2.5 is_host() . . . 531
10.16cl::sycl::is_wrapper< T > Struct Template Reference . . . 531
10.16.1 Detailed Description . . . 532
10.17cl::sycl::xilinx::partition::none Struct Reference . . . 532
10.17.1 Detailed Description . . . 532
10.17.2 Member Data Documentation . . . 533
10.17.2.1 partition_type . . . 533
10.18cl::sycl::detail::opencl_context Class Reference . . . 533
10.18.1 Detailed Description . . . 534

10.18.2 Constructor & Destructor Documentation . . . 535
10.18.2.1 opencl_context() . . . 535
10.18.2.2 ∼opencl_context() . . . 535
10.18.3 Member Function Documentation . . . 535
10.18.3.1 get() . . . 536
10.18.3.2 get_boost_compute() . . . 536
10.18.3.3 get_boost_queue() . . . 536
10.18.3.4 get_devices() . . . 537
10.18.3.5 get_platform() . . . 537
10.18.3.6 instance() . . . 538
10.18.3.7 is_host() . . . 539
10.18.4 Member Data Documentation . . . 539
10.18.4.1 c . . . 540
10.18.4.2 cache . . . 540
10.18.4.3 q . . . 540
10.19cl::sycl::detail::opencl_device Class Reference . . . 541
10.19.1 Detailed Description . . . 542
10.19.2 Constructor & Destructor Documentation . . . 542
10.19.2.1 opencl_device() . . . 542
10.19.2.2 ∼opencl_device() . . . 543
10.19.3 Member Function Documentation . . . 543
10.19.3.1 get() . . . 543
10.19.3.2 get_boost_compute() . . . 544
10.19.3.3 get_platform() . . . 544
10.19.3.4 has_extension() . . . 545
10.19.3.5 instance() . . . 545
10.19.3.6 is_accelerator() . . . 546
10.19.3.7 is_cpu() . . . 546
10.19.3.8 is_gpu() . . . 547
10.19.3.9 is_host() . . . 547

10.19.4 Member Data Documentation . . . 547
10.19.4.1 cache . . . 547
10.19.4.2 d . . . 548
10.20 cl::sycl::detail::opencl_kernel Class Reference . . . 548
10.20.1 Detailed Description . . . 550
10.20.2 Constructor & Destructor Documentation . . . 550
10.20.2.1 opencl_kernel() . . . 550
10.20.3 Member Function Documentation . . . 550
10.20.3.1 get() . . . 550
10.20.3.2 get_boost_compute() . . . 551
10.20.3.3 instance() . . . 551
10.20.3.4 single_task() . . . 552
10.20.3.5 TRISYCL_ParallelForKernel_RANGE() . . . 553
10.20.4 Member Data Documentation . . . 553
10.20.4.1 cache . . . 553
10.20.4.2 k . . . 554
10.21 cl::sycl::detail::opencl_queue Class Reference . . . 554
10.21.1 Detailed Description . . . 556
10.21.2 Constructor & Destructor Documentation . . . 556
10.21.2.1 opencl_queue() . . . 556
10.21.2.2 ~opencl_queue() . . . 556
10.21.3 Member Function Documentation . . . 557
10.21.3.1 get() . . . 557
10.21.3.2 get_boost_compute() . . . 557
10.21.3.3 get_context() . . . 557
10.21.3.4 get_device() . . . 558
10.21.3.5 instance() [1/2] . . . 558
10.21.3.6 instance() [2/2] . . . 559
10.21.3.7 is_host() . . . 559
10.21.4 Member Data Documentation . . . 559

10.21.4.1 cache . . . 560
10.21.4.2 q . . . 560
10.22 cl::sycl::info::param_traits< T, Param > Struct Template Reference . . . 560
10.22.1 Detailed Description . . . 560
10.23 cl::sycl::drt::code::program Struct Reference . . . 561
10.23.1 Detailed Description . . . 561
10.23.2 Constructor & Destructor Documentation . . . 561
10.23.2.1 program() . . . 562
10.23.3 Member Data Documentation . . . 562
10.23.3.1 binary . . . 562
10.23.3.2 binary_size . . . 562
10.23.3.3 p . . . 562
10.24 cl::sycl::detail::queue Struct Reference . . . 563
10.24.1 Detailed Description . . . 564
10.24.2 Constructor & Destructor Documentation . . . 564
10.24.2.1 queue() . . . 564
10.24.2.2 ~queue() . . . 565
10.24.3 Member Function Documentation . . . 565
10.24.3.1 get() . . . 565
10.24.3.2 get_boost_compute() . . . 566
10.24.3.3 get_context() . . . 566
10.24.3.4 get_device() . . . 567
10.24.3.5 is_host() . . . 567
10.24.3.6 kernel_end() . . . 568
10.24.3.7 kernel_start() . . . 568
10.24.3.8 wait_for_kernel_execution() . . . 569
10.24.4 Member Data Documentation . . . 569
10.24.4.1 finished . . . 569
10.24.4.2 finished_mutex . . . 569
10.24.4.3 running_kernels . . . 570

10.25 cl::sycl::detail::shared_ptr_implementation< Parent, Implementation > Struct Template Reference 570
10.25.1 Detailed Description . . . 571
10.25.2 Constructor & Destructor Documentation . . . 572
10.25.2.1 shared_ptr_implementation() [1/3] . . . 572
10.25.2.2 shared_ptr_implementation() [2/3] . . . 572
10.25.2.3 shared_ptr_implementation() [3/3] . . . 572
10.25.3 Member Function Documentation . . . 573
10.25.3.1 hash() . . . 573
10.25.3.2 operator<() . . . 574
10.25.3.3 operator==() . . . 574
10.25.4 Member Data Documentation . . . 574
10.25.4.1 implementation . . . 574
10.26 cl::sycl::detail::singleton< T > Struct Template Reference . . . 575
10.26.1 Detailed Description . . . 575
10.26.2 Member Function Documentation . . . 575
10.26.2.1 instance() . . . 575
10.27 cl::sycl::detail::task Struct Reference . . . 576
10.27.1 Detailed Description . . . 579
10.27.2 Constructor & Destructor Documentation . . . 579
10.27.2.1 task() . . . 579
10.27.3 Member Function Documentation . . . 579
10.27.3.1 add_buffer() . . . 579
10.27.3.2 add_postlude() . . . 580
10.27.3.3 add_prelude() . . . 580
10.27.3.4 get_compute_buffer() . . . 581
10.27.3.5 get_kernel() . . . 581
10.27.3.6 get_queue() . . . 582
10.27.3.7 notify_consumers() . . . 582
10.27.3.8 postlude() . . . 583
10.27.3.9 prelude() . . . 584

10.27.3.10 register_accessor() . . . 584
10.27.3.11 release_buffers() . . . 585
10.27.3.12 schedule() . . . 586
10.27.3.13 set_arg() [1/2] . . . 587
10.27.3.14 set_arg() [2/2] . . . 588
10.27.3.15 set_kernel() . . . 589
10.27.3.16 wait() . . . 590
10.27.3.17 wait_for_producers() . . . 590
10.27.4 Member Data Documentation . . . 590
10.27.4.1 accessors . . . 591
10.27.4.2 buffers_in_use . . . 591
10.27.4.3 epilogues . . . 591
10.27.4.4 execution_ended . . . 591
10.27.4.5 kernel . . . 592
10.27.4.6 owner_queue . . . 592
10.27.4.7 producer_tasks . . . 592
10.27.4.8 prologues . . . 592
10.27.4.9 ready . . . 592
10.27.4.10 ready_mutex . . . 592

11 File Documentation **593**

11.1 include/CL/sycl.hpp File Reference . . . 593
11.2 sycl.hpp . . . 594
11.3 include/CL/sycl/access.hpp File Reference . . . 595
11.4 access.hpp . . . 596
11.5 include/CL/sycl/accessor.hpp File Reference . . . 597
11.6 accessor.hpp . . . 598
11.7 include/CL/sycl/buffer/detail/accessor.hpp File Reference . . . 604
11.8 accessor.hpp . . . 605
11.9 include/CL/sycl/accessor/detail/accessor_base.hpp File Reference . . . 611
11.10 accessor_base.hpp . . . 612

11.11 include/CL/sycl/accessor/detail/local_accessor.hpp File Reference . . . 613
11.12 local_accessor.hpp . . . 615
11.13 include/CL/sycl/address_space/detail/address_space.hpp File Reference . . . 619
11.13.1 Detailed Description . . . 621
11.13.2 Macro Definition Documentation . . . 621
11.13.2.1 TRISYCL_CONSTANT_AS . . . 622
11.13.2.2 TRISYCL_GENERIC_AS . . . 622
11.13.2.3 TRISYCL_GLOBAL_AS . . . 622
11.13.2.4 TRISYCL_LOCAL_AS . . . 622
11.13.2.5 TRISYCL_PRIVATE_AS . . . 622
11.14 address_space.hpp . . . 623
11.15 include/CL/sycl/address_space.hpp File Reference . . . 627
11.15.1 Detailed Description . . . 629
11.16 address_space.hpp . . . 630
11.17 include/CL/sycl/allocator.hpp File Reference . . . 631
11.18 allocator.hpp . . . 633
11.19 include/CL/sycl/buffer/detail/buffer.hpp File Reference . . . 633
11.20 buffer.hpp . . . 635
11.21 include/CL/sycl/buffer.hpp File Reference . . . 640
11.22 buffer.hpp . . . 641
11.23 include/CL/sycl/buffer/detail/buffer_base.hpp File Reference . . . 648
11.24 buffer_base.hpp . . . 649
11.25 include/CL/sycl/buffer/detail/buffer_waiter.hpp File Reference . . . 653
11.26 buffer_waiter.hpp . . . 655
11.27 include/CL/sycl/buffer_allocator.hpp File Reference . . . 656
11.28 buffer_allocator.hpp . . . 657
11.29 include/CL/sycl/command_group/detail/task.hpp File Reference . . . 658
11.30 task.hpp . . . 659
11.31 include/CL/sycl/context/detail/context.hpp File Reference . . . 663
11.32 context.hpp . . . 664

11.33 include/CL/sycl/context.hpp File Reference . . . 665
11.34 context.hpp . . . 667
11.35 include/CL/sycl/info/context.hpp File Reference . . . 669
11.36 context.hpp . . . 671
11.37 include/CL/sycl/context/detail/host_context.hpp File Reference . . . 671
11.38 host_context.hpp . . . 673
11.39 include/CL/sycl/context/detail/opencl_context.hpp File Reference . . . 674
11.40 opencl_context.hpp . . . 676
11.41 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference . . . 677
11.41.1 Detailed Description . . . 679
11.42 array_tuple_helpers.hpp . . . 679
11.43 include/CL/sycl/detail/cache.hpp File Reference . . . 681
11.44 cache.hpp . . . 682
11.45 include/CL/sycl/detail/container_element_aspect.hpp File Reference . . . 684
11.46 container_element_aspect.hpp . . . 684
11.47 include/CL/sycl/detail/debug.hpp File Reference . . . 685
11.47.1 Macro Definition Documentation . . . 686
11.47.1.1 TRISYCL_DUMP . . . 687
11.47.1.2 TRISYCL_DUMP_T . . . 687
11.47.1.3 TRISYCL_INTERNAL_DUMP . . . 687
11.48 debug.hpp . . . 688
11.49 include/CL/sycl/detail/default_classes.hpp File Reference . . . 690
11.50 default_classes.hpp . . . 692
11.51 include/CL/sycl/detail/global_config.hpp File Reference . . . 693
11.51.1 Macro Definition Documentation . . . 695
11.51.1.1 TRISYCL_WEAK_ATTRIB_PREFIX . . . 695
11.51.1.2 TRISYCL_WEAK_ATTRIB_SUFFIX . . . 695
11.52 global_config.hpp . . . 696
11.53 include/CL/sycl/detail/instantiate_kernel.hpp File Reference . . . 697
11.54 instantiate_kernel.hpp . . . 698

11.55 include/CL/sycl/detail/linear_id.hpp File Reference . . . 700
11.56 linear_id.hpp . . . 701
11.57 include/CL/sycl/detail/shared_ptr_implementation.hpp File Reference . . . 701
11.58 shared_ptr_implementation.hpp . . . 703
11.59 include/CL/sycl/detail/singleton.hpp File Reference . . . 704
11.60 singleton.hpp . . . 705
11.61 include/CL/sycl/detail/small_array.hpp File Reference . . . 706
11.62 small_array.hpp . . . 708
11.63 include/CL/sycl/detail/SPIR/opencl_spir.h File Reference . . . 711
11.63.1 Macro Definition Documentation . . . 716
11.63.1.1 __kernel_exec . . . 717
11.63.1.2 as_char . . . 717
11.63.1.3 as_char16 . . . 717
11.63.1.4 as_char2 . . . 717
11.63.1.5 as_char3 . . . 717
11.63.1.6 as_char4 . . . 718
11.63.1.7 as_char8 . . . 718
11.63.1.8 as_double . . . 718
11.63.1.9 as_double16 . . . 718
11.63.1.10 as_double2 . . . 718
11.63.1.11 as_double3 . . . 718
11.63.1.12 as_double4 . . . 719
11.63.1.13 as_double8 . . . 719
11.63.1.14 as_float . . . 719
11.63.1.15 as_float16 . . . 719
11.63.1.16 as_float2 . . . 719
11.63.1.17 as_float3 . . . 719
11.63.1.18 as_float4 . . . 720
11.63.1.19 as_float8 . . . 720
11.63.1.20 as_half . . . 720

11.63.1.21as_half16 720
11.63.1.22as_half2 720
11.63.1.23as_half3 720
11.63.1.24as_half4 721
11.63.1.25as_half8 721
11.63.1.26as_int 721
11.63.1.27as_int16 721
11.63.1.28as_int2 721
11.63.1.29as_int3 721
11.63.1.30as_int4 722
11.63.1.31as_int8 722
11.63.1.32as_long 722
11.63.1.33as_long16 722
11.63.1.34as_long2 722
11.63.1.35as_long3 722
11.63.1.36as_long4 723
11.63.1.37as_long8 723
11.63.1.38as_short 723
11.63.1.39as_short16 723
11.63.1.40as_short2 723
11.63.1.41as_short3 723
11.63.1.42as_short4 724
11.63.1.43as_short8 724
11.63.1.44as_uchar 724
11.63.1.45as_uchar16 724
11.63.1.46as_uchar2 724
11.63.1.47as_uchar3 724
11.63.1.48as_uchar4 725
11.63.1.49as_uchar8 725
11.63.1.50as_uint 725

11.63.1.51as_uint16 725
11.63.1.52as_uint2 725
11.63.1.53as_uint3 725
11.63.1.54as_uint4 726
11.63.1.55as_uint8 726
11.63.1.56as_ulong 726
11.63.1.57as_ulong16 726
11.63.1.58as_ulong2 726
11.63.1.59as_ulong3 726
11.63.1.60as_ulong4 727
11.63.1.61as_ulong8 727
11.63.1.62as_ushort 727
11.63.1.63as_ushort16 727
11.63.1.64as_ushort2 727
11.63.1.65as_ushort3 727
11.63.1.66as_ushort4 728
11.63.1.67as_ushort8 728
11.63.1.68CHAR_BIT 728
11.63.1.69CHAR_MAX 728
11.63.1.70CHAR_MIN 728
11.63.1.71CLK_A 728
11.63.1.72CLK_ADDRESS_CLAMP 729
11.63.1.73CLK_ADDRESS_CLAMP_TO_EDGE 729
11.63.1.74CLK_ADDRESS_MIRRORED_REPEAT 729
11.63.1.75CLK_ADDRESS_NONE 729
11.63.1.76CLK_ADDRESS_REPEAT 729
11.63.1.77CLK_ARGB 730
11.63.1.78CLK_BGRA 730
11.63.1.79CLK_DEPTH 730
11.63.1.80CLK_DEPTH_STENCIL 730

11.63.1.81 CLK_FILTER_LINEAR 730
11.63.1.82 CLK_FILTER_NEAREST 730
11.63.1.83 CLK_FLOAT 731
11.63.1.84 CLK_GLOBAL_MEM_FENCE 731
11.63.1.85 CLK_HALF_FLOAT 731
11.63.1.86 CLK_INTENSITY 731
11.63.1.87 CLK_LOCAL_MEM_FENCE 731
11.63.1.88 CLK_LUMINANCE 731
11.63.1.89 CLK_NORMALIZED_COORDS_FALSE 732
11.63.1.90 CLK_NORMALIZED_COORDS_TRUE 732
11.63.1.91 CLK_R 732
11.63.1.92 CLK_RA 732
11.63.1.93 CLK_RG 732
11.63.1.94 CLK_RGB 733
11.63.1.95 CLK_RGBA 733
11.63.1.96 CLK_RGBx 733
11.63.1.97 CLK_RGx 733
11.63.1.98 CLK_Rx 733
11.63.1.99 CLK_SIGNED_INT16 733
11.63.1.100 CLK_SIGNED_INT32 734
11.63.1.101 CLK_SIGNED_INT8 734
11.63.1.102 CLK_SNORM_INT16 734
11.63.1.103 CLK_SNORM_INT8 734
11.63.1.104 CLK_UNORM_INT16 734
11.63.1.105 CLK_UNORM_INT24 734
11.63.1.106 CLK_UNORM_INT8 735
11.63.1.107 CLK_UNORM_INT_101010 735
11.63.1.108 CLK_UNORM_SHORT_555 735
11.63.1.109 CLK_UNORM_SHORT_565 735
11.63.1.110 CLK_UNSIGNED_INT16 735

11.63.1.111 CLK_UNSIGNED_INT32 735
11.63.1.112 CLK_UNSIGNED_INT8 736
11.63.1.113 const_func 736
11.63.1.114 DBL_DIG [1/2] 736
11.63.1.115 DBL_DIG [2/2] 736
11.63.1.116 DBL_EPSILON [1/2] 736
11.63.1.117 DBL_EPSILON [2/2] 737
11.63.1.118 DBL_MANT_DIG [1/2] 737
11.63.1.119 DBL_MANT_DIG [2/2] 737
11.63.1.120 DBL_MAX [1/2] 737
11.63.1.121 DBL_MAX [2/2] 737
11.63.1.122 DBL_MAX_10_EXP [1/2] 737
11.63.1.123 DBL_MAX_10_EXP [2/2] 738
11.63.1.124 DBL_MAX_EXP [1/2] 738
11.63.1.125 DBL_MAX_EXP [2/2] 738
11.63.1.126 DBL_MIN [1/2] 738
11.63.1.127 DBL_MIN [2/2] 738
11.63.1.128 DBL_MIN_10_EXP [1/2] 738
11.63.1.129 DBL_MIN_10_EXP [2/2] 739
11.63.1.130 DBL_MIN_EXP [1/2] 739
11.63.1.131 DBL_MIN_EXP [2/2] 739
11.63.1.132 DBL_RADIX 739
11.63.1.133 FLT_DIG 739
11.63.1.134 FLT_EPSILON 739
11.63.1.135 FLT_MANT_DIG 740
11.63.1.136 FLT_MAX 740
11.63.1.137 FLT_MAX_10_EXP 740
11.63.1.138 FLT_MAX_EXP 740
11.63.1.139 FLT_MIN 740
11.63.1.140 FLT_MIN_10_EXP 740

11.63.1.141 FLT_MIN_EXP 741
11.63.1.142 FLT_RADIX 741
11.63.1.143 FP_ILOGB0 741
11.63.1.144 FP_ILOGBNAN 741
11.63.1.145 HUGE_VAL 741
11.63.1.146 HUGE_VALF 742
11.63.1.147 INFINITY 742
11.63.1.148 INT_MAX 742
11.63.1.149 INT_MIN 742
11.63.1.150 kernel_exec 742
11.63.1.151 LONG_MAX 743
11.63.1.152 LONG_MIN 743
11.63.1.153 M_1_PI 743
11.63.1.154 M_1_PI_F 743
11.63.1.155 M_2_PI 743
11.63.1.156 M_2_PI_F 743
11.63.1.157 M_2_SQRTPI 744
11.63.1.158 M_2_SQRTPI_F 744
11.63.1.159 M_E 744
11.63.1.160 M_E_F 744
11.63.1.161 M_LN10 744
11.63.1.162 M_LN10_F 744
11.63.1.163 M_LN2 745
11.63.1.164 M_LN2_F 745
11.63.1.165 M_LOG10E 745
11.63.1.166 M_LOG10E_F 745
11.63.1.167 M_LOG2E 745
11.63.1.168 M_LOG2E_F 745
11.63.1.169 M_PI 746
11.63.1.170 M_PI_2 746

11.63.1.170 M_PI_2_F . . . 746
11.63.1.171 M_PI_4 . . . 746
11.63.1.172 M_PI_4_F . . . 746
11.63.1.173 M_PI_F . . . 746
11.63.1.174 M_SQRT1_2 . . . 747
11.63.1.175 M_SQRT1_2_F . . . 747
11.63.1.176 M_SQRT2 . . . 747
11.63.1.177 M_SQRT2_F . . . 747
11.63.1.178 MAXFLOAT . . . 747
11.63.1.179 NAN . . . 747
11.63.1.180 readonly . . . 748
11.63.1.181 SCHAR_MAX . . . 748
11.63.1.182 SCHAR_MIN . . . 748
11.63.1.183 SHRT_MAX . . . 748
11.63.1.184 SHRT_MIN . . . 748
11.63.1.185 UCHAR_MAX . . . 748
11.63.1.186 UINT_MAX . . . 749
11.63.1.187 ULONG_MAX . . . 749
11.63.1.188 USHRT_MAX . . . 749
11.63.2 Typedef Documentation . . . 749
11.63.2.1 __attribute__ . . . 749
11.63.2.2 cl_mem_fence_flags . . . 760
11.63.2.3 intptr_t . . . 760
11.63.2.4 uchar . . . 761
11.63.2.5 uint . . . 761
11.63.2.6 uintptr_t . . . 761
11.63.2.7 ulong . . . 761
11.63.2.8 ushort . . . 761
11.63.3 Function Documentation . . . 761
11.63.3.1 __attribute__() . . . 762

11.63.3.2 printf() . . . 772

11.63.4 Variable Documentation . . . 772

11.63.4.1 a . . . 773

11.63.4.2 b . . . 773

11.63.4.3 c . . . 773

11.63.4.4 cmp . . . 773

11.63.4.5 color . . . 773

11.63.4.6 coord . . . 773

11.63.4.7 cosval . . . 774

11.63.4.8 dst_stride . . . 774

11.63.4.9 edge1 . . . 774

11.63.4.10 event . . . 774

11.63.4.11 event_list . . . 774

11.63.4.12 exp . . . 774

11.63.4.13 i . . . 775

11.63.4.14 iptr . . . 775

11.63.4.15 o . . . 775

11.63.4.16 mask . . . 775

11.63.4.17 maxval . . . 775

11.63.4.18 minval . . . 776

11.63.4.19 n . . . 776

11.63.4.20 num_elements . . . 776

11.63.4.21 offset . . . 776

11.63.4.22 p . . . 776

11.63.4.23 p1 . . . 777

11.63.4.24 quo . . . 777

11.63.4.25 sampler . . . 777

11.63.4.26 signp . . . 777

11.63.4.27 src . . . 777

11.63.4.28 src_stride . . . 777

11.63.4.29val . . . 778
11.63.4.30x . . . 778
11.63.4.31y . . . 778
11.63.4.32z . . . 778
11.64opencl_spir.h . . . 778
11.65include/CL/sycl/detail/unimplemented.hpp File Reference . . . 982
11.66unimplemented.hpp . . . 983
11.67include/CL/sycl/device/detail/device.hpp File Reference . . . 984
11.68device.hpp . . . 985
11.69include/CL/sycl/device.hpp File Reference . . . 987
11.70device.hpp . . . 988
11.71include/CL/sycl/info/device.hpp File Reference . . . 992
11.72device.hpp . . . 995
11.73include/CL/sycl/device/detail/device_tail.hpp File Reference . . . 997
11.74device_tail.hpp . . . 998
11.75include/CL/sycl/device/detail/host_device.hpp File Reference . . . 998
11.76host_device.hpp . . . 1000
11.77include/CL/sycl/device/detail/opencl_device.hpp File Reference . . . 1001
11.78opencl_device.hpp . . . 1002
11.79include/CL/sycl/device_runtime.hpp File Reference . . . 1004
11.80device_runtime.hpp . . . 1006
11.81include/CL/sycl/device_selector.hpp File Reference . . . 1009
11.82device_selector.hpp . . . 1010
11.83include/CL/sycl/device_selector/detail/device_selector_tail.hpp File Reference . . . 1011
11.84device_selector_tail.hpp . . . 1013
11.85include/CL/sycl/error_handler.hpp File Reference . . . 1015
11.86error_handler.hpp . . . 1016
11.87include/CL/sycl/event.hpp File Reference . . . 1017
11.88event.hpp . . . 1018
11.89include/CL/sycl/exception.hpp File Reference . . . 1018

11.90 exception.hpp . . . 1021
11.91 include/CL/sycl/group.hpp File Reference . . . 1023
11.92 group.hpp . . . 1025
11.93 include/CL/sycl/handler.hpp File Reference . . . 1027
11.93.1 Macro Definition Documentation . . . 1029
11.93.1.1 TRISYCL_parallel_for_functor_GLOBAL . . . 1029
11.93.1.2 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET . . . 1029
11.93.1.3 TRISYCL_ParallelForKernel_RANGE . . . 1030
11.94 handler.hpp . . . 1030
11.95 include/CL/sycl/handler_event.hpp File Reference . . . 1037
11.96 handler_event.hpp . . . 1037
11.97 include/CL/sycl/id.hpp File Reference . . . 1038
11.98 id.hpp . . . 1039
11.99 include/CL/sycl/image.hpp File Reference . . . 1040
11.99.1 Detailed Description . . . 1041
11.100 image.hpp . . . 1041
11.101 include/CL/sycl/info/param_traits.hpp File Reference . . . 1042
11.101.1 Macro Definition Documentation . . . 1043
11.101.1.1 TRISYCL_INFO_PARAM_TRAITS . . . 1043
11.101.1.2 TRISYCL_INFO_PARAM_TRAITS_ANY_T . . . 1043
11.102 param_traits.hpp . . . 1044
11.103 include/CL/sycl/info/platform.hpp File Reference . . . 1044
11.104 platform.hpp . . . 1046
11.105 include/CL/sycl/platform/detail/platform.hpp File Reference . . . 1047
11.106 platform.hpp . . . 1049
11.107 include/CL/sycl/platform.hpp File Reference . . . 1050
11.108 platform.hpp . . . 1051
11.109 include/CL/sycl/item.hpp File Reference . . . 1053
11.110 item.hpp . . . 1055
11.111 include/CL/sycl/kernel/detail/kernel.hpp File Reference . . . 1056

11.111.1 Macro Definition Documentation . . . 1058
11.111.1.1 TRISYCL_ParallelForKernel_RANGE . . . 1058
11.112 kernel.hpp . . . 1058
11.113 include/CL/sycl/kernel.hpp File Reference . . . 1059
11.114 kernel.hpp . . . 1061
11.115 include/CL/sycl/kernel/detail/opencl_kernel.hpp File Reference . . . 1062
11.115.1 Macro Definition Documentation . . . 1064
11.115.1.1 TRISYCL_ParallelForKernel_RANGE . . . 1064
11.116 opencl_kernel.hpp . . . 1065
11.117 include/CL/sycl/math.hpp File Reference . . . 1066
11.117.1 Detailed Description . . . 1068
11.117.2 Macro Definition Documentation . . . 1068
11.117.2.1 TRISYCL_MATH_WRAP . . . 1068
11.117.2.2 TRISYCL_MATH_WRAP2 . . . 1069
11.117.2.3 TRISYCL_MATH_WRAP2s . . . 1069
11.117.2.4 TRISYCL_MATH_WRAP3 . . . 1069
11.117.2.5 TRISYCL_MATH_WRAP3s . . . 1070
11.117.2.6 TRISYCL_MATH_WRAP3ss . . . 1070
11.118 math.hpp . . . 1070
11.119 include/CL/sycl/nd_item.hpp File Reference . . . 1072
11.120 nd_item.hpp . . . 1074
11.121 include/CL/sycl/nd_range.hpp File Reference . . . 1077
11.122 nd_range.hpp . . . 1078
11.123 include/CL/sycl/opencl_types.hpp File Reference . . . 1079
11.123.1 Detailed Description . . . 1081
11.123.2 Macro Definition Documentation . . . 1081
11.123.2.1 TRISYCL_BOOST_COMPUTE_NAME . . . 1081
11.123.2.2 TRISYCL_DECLARE_CL_TYPES . . . 1081
11.123.2.3 TRISYCL_DEFINE_TYPES . . . 1082
11.123.2.4 TRISYCL_H_DEFINE_TYPE . . . 1082

11.123.2.5 TRISYCL_IS_WRAPPER_TRAIT . . . 1082
11.123.2.6 TRISYCL_SCALAR_TYPES . . . 1083
11.123.2.7 TRISYCL_SIZED_NAME . . . 1083
11.123.2.8 TRISYCL_TYPE_ACTUAL_NAME . . . 1083
11.123.2.9 TRISYCL_TYPE_CL_NAME . . . 1083
11.123.2.10 TRISYCL_TYPE_NAME . . . 1084
11.123.2.11 TRISYCL_TYPEDEF_TYPE . . . 1084
11.123.2.12 TRISYCL_WRAPPER_CLASS_2 . . . 1084
11.123.2.13 TRISYCL_WRAPPER_CLASS_3 . . . 1084
11.123.2.14 TRISYCL_WRAPPER_CLASS_4 . . . 1085
11.124 opencl_types.hpp . . . 1085
11.125 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference . . . 1087
11.125.1 Detailed Description . . . 1089
11.126 parallelism.hpp . . . 1090
11.127 include/CL/sycl/parallelism.hpp File Reference . . . 1094
11.127.1 Detailed Description . . . 1095
11.128 parallelism.hpp . . . 1096
11.129 include/CL/sycl/pipe/detail/pipe.hpp File Reference . . . 1096
11.130 pipe.hpp . . . 1098
11.131 include/CL/sycl/pipe.hpp File Reference . . . 1103
11.132 pipe.hpp . . . 1104
11.133 include/CL/sycl/pipe/detail/pipe_accessor.hpp File Reference . . . 1106
11.134 pipe_accessor.hpp . . . 1107
11.135 include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp File Reference . . . 1110
11.136 pipe_reservation.hpp . . . 1112
11.137 include/CL/sycl/pipe_reservation.hpp File Reference . . . 1114
11.138 pipe_reservation.hpp . . . 1115
11.139 include/CL/sycl/platform/detail/host_platform.hpp File Reference . . . 1117
11.140 host_platform.hpp . . . 1119
11.141 include/CL/sycl/platform/detail/host_platform_tail.hpp File Reference . . . 1121

11.142 host_platform_tail.hpp . . . 1121
11.143 include/CL/sycl/platform/detail/opencl_platform.hpp File Reference . . . 1122
11.144 opencl_platform.hpp . . . 1123
11.145 include/CL/sycl/platform/detail/opencl_platform_tail.hpp File Reference . . . 1125
11.146 opencl_platform_tail.hpp . . . 1126
11.147 include/CL/sycl/queue/detail/host_queue.hpp File Reference . . . 1126
11.148 host_queue.hpp . . . 1128
11.149 include/CL/sycl/queue/detail/opencl_queue.hpp File Reference . . . 1129
11.150 opencl_queue.hpp . . . 1130
11.151 include/CL/sycl/queue/detail/queue.hpp File Reference . . . 1131
11.152 queue.hpp . . . 1133
11.153 include/CL/sycl/queue.hpp File Reference . . . 1134
11.154 queue.hpp . . . 1136
11.155 include/CL/sycl/range.hpp File Reference . . . 1140
11.156 range.hpp . . . 1142
11.157 include/CL/sycl/static_pipe.hpp File Reference . . . 1143
11.158 static_pipe.hpp . . . 1144
11.159 include/CL/sycl/vec.hpp File Reference . . . 1145
11.159.1 Detailed Description . . . 1147
11.160 vec.hpp . . . 1147
11.161 include/CL/sycl/vendor/Xilinx/opt_decorate_func.hpp File Reference . . . 1150
11.162 opt_decorate_func.hpp . . . 1150
11.163 include/CL/sycl/vendor/Xilinx/partition_array.hpp File Reference . . . 1151
11.164 partition_array.hpp . . . 1153
11.165 include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp File Reference . . . 1156
11.166 ssdm_inst.hpp . . . 1156

Index 1159

Chapter 1

Main Page

This is the main OpenCL SYCL C++ header file to experiment with the OpenCL CL provisional specification.For more information about OpenCL SYCL: http://www.khronos.org/sycl/

For more information on this project and to access to the source of this file, look at https://github.com/triSYCL/triSYCL

The Doxygen version of the implementation itself is in http://Xilinx.github.io/triSYCL/Doxygen/triSYCL/html and http://Xilinx.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf

Ronan at keryell dot FR

Copyright 2014–2015 Advanced Micro Devices, Inc.

Copyright 2015–2017 Xilinx, Inc.

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Chapter 2

Todo List

File address_space.hpp

Add the alias ..._ptr<T> = ...<T *>

Namespace cl::sycl::access

This values should be normalized to allow separate compilation with different implementations?

Class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

Implement it for images according so section 3.3.4.5

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler)

Add template allocator type in all the accessor constructors in the specification or just use a more opaque Buffer type?

fix specification where access mode should be target instead

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::begin () const

Add these functions to the specification

The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

try to solve it by using some enable_if on array constness?

The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Factor out these in a template helper

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::dimensionality

in the specification: store the dimension for user request

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_count () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_pointer () const

Should it be named data() instead?

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_range () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_size () const

It is incompatible with buffer get_size() in the spec

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator* ()

Add in the specification

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator* () const

Add in the specification?

Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (nd_item< dimensionality > index)

Add in the specification because used by HPC-GPU slide 22

Member cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (nd_item< dimensionality > index) const

Add in the specification because used by HPC-GPU slide 22

Class cl::sycl::buffer< T, Dimensions, Allocator >

There is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Finish allocator implementation

Think about the need of an allocator when constructing a buffer from other buffers

Update the specification to have a non-const allocator for const buffer? Or do we rely on rebind_alloc<T>. But does this work with astate-full allocator?

Add constructors from arrays so that in C++17 the range and type can be infered from the constructor

Add constructors from array_ref

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (const T *host_data, const range< Dimensions > &r, Allocator allocator={})

Actually this is redundant.

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (shared_ptr_class< T > &host_data, const range< Dimensions > &buffer_range, cl::sycl::mutex_class &m, Allocator allocator={})

update the specification to replace the pointer by a reference and provide the constructor with and without a mutex

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (shared_ptr_class< T > host_data, const range< Dimensions > &buffer_range, Allocator allocator={})

add this mutex-less constructor to the specification

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (InputIterator start_iterator, InputIterator end↩_iterator, Allocator allocator={})

Implement the copy back at buffer destruction

Generalize this for n-D and provide column-major and row-major initialization

a reason to have this nD is that set_final_data(weak_ptr_class<T> & finalData) is actually doing this linearization anyway

Allow read-only buffer construction too

update the specification to deal with forward iterators instead and rewrite back only when it is non const and output iterator at least

Allow initialization from ranges and collections à la STL

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (buffer< T, Dimensions, Allocator > &b, const id< Dimensions > &base_index, const range< Dimensions > &sub_range, Allocator allocator={})

To be implemented

Update the specification to replace index by id

Member cl::sycl::buffer< T, Dimensions, Allocator >::buffer (cl_mem mem_object, queue from_queue, event available_event={}, Allocator allocator={})

To be implemented

Improve the specification to allow CLHPP objects too

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_access (handler &command_group_handler)

Do we need for an accessor to increase the reference count of a buffer object? It does make more sense for a host-side accessor.

Implement the modes and targets

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_access ()

Implement the modes

More elegant solution

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_range () const

rename to the equivalent from array_ref proposals? Such as size() in http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html

Member cl::sycl::buffer< T, Dimensions, Allocator >::get_size () const

rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf it is named bytes() for example

Member cl::sycl::buffer< T, Dimensions, Allocator >::is_read_only () const

Add to specification

Member cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (shared_ptr_class< T > finalData)

Update the API to take finalData by value instead of by reference. This way we can have an implicit conversion possible at the API call from a shared_ptr<>, avoiding an explicit weak_ptr<> creation

figure out how set_final_data() interact with the other way to write back some data or with some data sharing with the host that can not be undone

Member cl::sycl::buffer< T, Dimensions, Allocator >::use_count () const

Add to the specification, useful for validation

Class cl::sycl::context

The implementation is quite minimal for now.

Member cl::sycl::context::get_devices () const

To be implemented

Member cl::sycl::context::get_info () const

To be implemented

Member cl::sycl::context::get_platform ()

To be implemented

Class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >

Use the access::mode

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::accessor (const range< Dimensions > &allocation_size, handler &command_group_handler)

fix the specification to rename target that shadows template param

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::begin () const

Add these functions to the specification

The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

try to solve it by using some enable_if on array constness?

The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Factor out these in a template helper

Do we need this in detail::accessor too or only in accessor?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::dimensionality

in the specification: store the dimension for user request

Use another name, such as from C++17 committee discussions.

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_count () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_range () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_size () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::is_read_access () const

Strangely, it is not really constexpr because it is not a static method...

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::is_write_access () const

Strangely, it is not really constexpr because it is not a static method...

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::iterator

Add iterators to accessors in the specification

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator* ()

Add in the specification

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator* () const

Add in the specification?

Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (nd_item< dimensionality > index)

Add in the specification because used by HPC-GPU slide 22

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (nd_item< dimensionality > index) const

Add in the specification because used by HPC-GPU slide 22

Member cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::value_type

in the specification: store the types for user request as STL or C++AMP

Class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

Use the access::mode

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (std::shared_ptr< detail↩::buffer< T, Dimensions >> target_buffer)

fix the specification to rename target that shadows template parm

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (std::shared_ptr< detail↩::buffer< T, Dimensions >> target_buffer, handler &command_group_handler)

fix the specification to rename target that shadows template parm

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin () const

Add these functions to the specification

The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

try to solve it by using some enable_if on array constness?

The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Factor out these in a template helper

Do we need this in detail::accessor too or only in accessor?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::dimensionality

in the specification: store the dimension for user request

Use another name, such as from C++17 committee discussions.

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_pointer () const

Implement the various pointer address spaces

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_range () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size () const

Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_read_access () const

Strangely, it is not really constexpr because it is not a static method...

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access () const

Strangely, it is not really constexpr because it is not a static method...

to move in the access::mode enum class and add to the specification ?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::iterator

Add iterators to accessors in the specification

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* ()

Add in the specification

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* () const

Add in the specification?

Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (nd_item< dimensionality > index)

Add in the specification because used by HPC-GPU slide 22

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (nd_item< dimensionality > index) const

Add in the specification because used by HPC-GPU slide 22

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor ()

Double-check with the C++ committee on this issue.

Member cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::value_type

in the specification: store the types for user request as STL or C++AMP

Member cl::sycl::detail::address_space_array< T, AS >::address_space_array (std::initializer_list< std::remove_extent_t< T >> list)

Extend to more than 1 dimension

Class cl::sycl::detail::address_space_base< T, AS >

Verify/improve to deal with const/volatile?

Member cl::sycl::detail::address_space_base< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::address_space_base< T, AS >::type

Add to the specification

Class cl::sycl::detail::address_space_fundamental< T, AS >

Verify/improve to deal with const/volatile?

Class cl::sycl::detail::address_space_object< T, AS >

Verify/improve to deal with const/volatile?

what about T having some final methods?

Member cl::sycl::detail::address_space_object< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::address_space_variable< T, AS >::opencl_type

Add to the specification

Member cl::sycl::detail::buffer< T, Dimensions >::alloc

Implement user-provided allocator

Member cl::sycl::detail::buffer< T, Dimensions >::buffer (const T *host_data, const range< Dimensions > &r)

Clarify the semantics in the spec. What happens if the host change the host_data after buffer creation?

Member cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state (cl::sycl::context ctx, access::mode mode, size_t size, DataType *data, std::enable_if_t<!std::is_const< BaseType > ::value > *=0)

Use `if constexpr` when it is available with C++17

Member cl::sycl::detail::buffer< T, Dimensions >::get_destructor_future ()

Make the function private again

Member cl::sycl::detail::buffer< T, Dimensions >::get_size () const

rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf it is named bytes() for example

Member cl::sycl::detail::buffer< T, Dimensions >::~buffer ()

To implement and deal with reference counting buffer(buffer<T, Dimensions> b, index<Dimensions> base_index, range<Dimensions> sub_range)

Allow CLHPP objects too?

Member cl::sycl::detail::buffer_add_to_task (BufferDetail buf, handler *command_group_handler, bool is_write_mode)

To remove with some refactoring

Member cl::sycl::detail::buffer_base::buffer_base ()

Use lazy allocation for the context tracking set

Member cl::sycl::detail::context::get_devices () const =0

virtual cannot be templated template <info::context Param> typename info::param_traits<info::context, Param>::type get_info() const = 0;

Member cl::sycl::detail::device::has_extension (const string_class &extension) const =0

virtual cannot be templated template <typename t>=""> virtual T get_info(info::device param) const = 0;

Member cl::sycl::detail::host_context::get_devices () const override

To be implemented

Class cl::sycl::detail::host_device

The implementation is quite minimal for now. :-)

Member cl::sycl::detail::host_device::get_platform () const override

To be implemented

Member cl::sycl::detail::host_device::has_extension (const string_class &extension) const override

To be implemented

Member cl::sycl::detail::host_platform::has_extension (const string_class &extension) const override

To be implemented

Member cl::sycl::detail::opencl_context::get_devices () const override

To be implemented

Member cl::sycl::detail::opencl_context::get_platform () const override

To be implemented

Member cl::sycl::detail::opencl_device::has_extension (const string_class &extension) const override

To be implemented

Member cl::sycl::detail::opencl_kernel::get () const override

Improve the spec to deprecate C OpenCL host API and move to C++ instead to avoid this ugly ownership management

Test error and throw. Externalize this feature in Boost.Compute?

Member cl::sycl::detail::opencl_kernel::single_task (std::shared_ptr< detail::task > task, std::shared_ptr< detail::queue > q) override

Remove either task or q

Member cl::sycl::detail::opencl_queue::instance (const cl::sycl::device &d)

Check with SYCL committee what is the expected behaviour here about the context. Is this a new context everytime, or always the same for a given device?

Member cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

Add an OpenMP implementation

Deal with incomplete work-groups

Implement with parallel_for_workgroup()/parallel_for_workitem()

Member cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

Better type the functor

Member cl::sycl::detail::pipe< T >::write (const T &value, bool blocking=false)

provide a && version

Member cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor (const std::shared_ptr< detail::pipe< T >> &p, handler &command_group_handler)

Use pipe_exception instead

Member cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::write (const value_type &value) const

provide a && version

Member cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity ()

Throw exception instead

Member cl::sycl::detail::pipe_reservation< PipeAccessor >::commit ()

Add to the specification that for simplicity a reservation can be commited several times but only the first one is taken into account

Member cl::sycl::detail::queue::~queue ()

Update according spec since queue destruction is non blocking

Member cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::operator< (const Parent &other) const

Add this to the spec

Member cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (BasicType e)

Add to the specification of the range, id...

Member cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (BasicType e)

Add to the specification of the range, id...

Member cl::sycl::detail::task::buffers_in_use

Use a set to check that some buffers are not used many times at least on writing

Member cl::sycl::detail::task::get_kernel ()

Specify this error in the spec

Member cl::sycl::detail::task::schedule (std::function< void(void)> f)

This is an issue if there is an exception in the kernel

Member cl::sycl::device::device (const device_selector &ds)

Make it non-explicit in the specification?

Member cl::sycl::device::get_info () const

Member cl::sycl::device::get_info (info::device param) const

Member cl::sycl::device::type () const

Present in Boost.Compute, to be added to the specification

Member cl::sycl::device_selector::select_device () const

Remove this from specification

Class cl::sycl::device_type_selector

To be added to the specification

Class cl::sycl::device_typename_selector< DeviceType >

To be added to the specification

Member cl::sycl::drt::code::program::p

Deal with several programs

Member cl::sycl::drt::kernel_IR

use std::byte when moving to C++17

Member cl::sycl::error_handler::default_handler

add this concept to the specification?

Member cl::sycl::error_handler::report_error (exception &error)=0

Add "virtual void" to the specification

Class cl::sycl::exception_list

Do we need to define it in SYCL or can we rely on plain C++17 one?

Member cl::sycl::exception_ptr

Do we need this instead of reusing directly the one from C++11?

Member cl::sycl::group< Dimensions >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::group< Dimensions >::get_group_range () const

Fix this comment and the specification

Member cl::sycl::group< Dimensions >::get_local_range () const

Add to the specification

Member cl::sycl::group< Dimensions >::get_local_range (int dimension) const

Add to the specification

Member cl::sycl::group< Dimensions >::get_nd_range () const

Also provide this access to the current nd_range

Member cl::sycl::group< Dimensions >::get_offset (int dimension) const

Add to the specification

Member cl::sycl::group< Dimensions >::get_offset () const

Add to the specification

Member cl::sycl::group< Dimensions >::group (const id< Dimensions > &i, const nd_range< Dimensions > &ndr)

This should be private somehow, but it is used by the validation infrastructure

Member cl::sycl::group< Dimensions >::group ()=default

Make most of them protected, reserved to implementation

Member cl::sycl::group< Dimensions >::group (const nd_range< Dimensions > &ndr)

This should be private since it is only used by the triSYCL implementation

Member cl::sycl::group< Dimensions >::operator[] (int dimension)

In this implementation it is not const because the group<> is written in the parallel_for iterators. To fix according to the specification

Member cl::sycl::group< Dimensions >::parallel_for_work_item (std::function< void(nd_item< dimensionality >)> f) const

Add this method in the specification

Member cl::sycl::group< Dimensions >::parallel_for_work_item (std::function< void(item< dimensionality >)> f) const

Add this method in the specification

Member cl::sycl::handler::set_arg (int arg_index, accessor< DataType, Dimensions, Mode, Target > &&acc_obj)

Update the specification to use a ref && to the accessor instead?

It is not that clean to have set_arg() associated to a command handler. Rethink the specification?

It seems more logical to have these methods on kernel instead

Member cl::sycl::handler::set_args (Ts &&... args)

Update the specification to add this function according to https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978 proposal

Member cl::sycl::handler::single_task (kernel sycl_kernel)

Add in the spec a version taking a kernel and a functor, to have host fall-back

Class cl::sycl::image< Dimensions >

implement image

Member cl::sycl::info::context

Should be unsigned int to be consistent with others?

Member cl::sycl::info::device

Should be unsigned int?

Member cl::sycl::info::device_type

To be moved in the specification from platform to device

Add opencl to the specification

there is no accelerator_selector and custom_accelerator

Member cl::sycl::info::queue

unsigned int?

To be implemented

To be implemented

Member cl::sycl::item< Dimensions >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::item< Dimensions >::item ()=default

Make most of them protected, reserved to implementation

Member cl::sycl::item< Dimensions >::set (id< Dimensions > Index)

Move to private and add friends

Class cl::sycl::kernel

To be implemented

Check specification

Member cl::sycl::make_multi (multi_ptr< T, AS > pointer)

Implement the case with a plain pointer

Member cl::sycl::map_allocator

: implement and clarify the specification. It looks like it is not really an allocator according the current spec

Member cl::sycl::nd_item< Dimensions >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::nd_item< Dimensions >::get_item () const

Add to the specification

Member cl::sycl::nd_item< Dimensions >::nd_item (id< Dimensions > global_index, nd_range< Dimensions > ndr)

This is for validation purpose. Hide this to the programmer somehow

Member cl::sycl::nd_item< Dimensions >::nd_item ()=default

Make most of them protected, reserved to implementation

Member cl::sycl::nd_item< Dimensions >::nd_item (nd_range< Dimensions > ndr)

This is for the triSYCL implementation which is expected to call set_global() and set_local() later. This should be hidden to the user.

Class cl::sycl::nd_range< Dimensions >

add copy constructors in the specification

Member cl::sycl::nd_range< Dimensions >::dimensionality

add this Boost::multi_array or STL concept to the specification?

Member cl::sycl::nd_range< Dimensions >::get_offset () const

get_offset() is lacking in the specification

Class cl::sycl::non_cl_error

Add to the specification

Clean implementation

Exceptions are named error in C++

Member cl::sycl::parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

To be implemented

Deprecate this function in the specification to use instead the group method

Member cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (detail::pipe_reservation< accessor_detail > &&pr)

Make it private and add required friends

Class cl::sycl::platform

triSYCL Implementation

Member cl::sycl::platform::get () const

Define a SYCL exception for this

Member cl::sycl::platform::get_info (info::platform param) const

Add to the specification

Class cl::sycl::queue

The implementation is quite minimal for now. :-)

All the queue methods should return a queue& instead of void to it is possible to chain opoerations

Member cl::sycl::queue::queue (const boost::compute::command_queue &q, async_handler ah=nullptr)

Deal with handler

Member cl::sycl::queue::queue ()

Check with the specification if it is the host queue or the one related to the default device selector.

Member cl::sycl::queue::submit (Handler_Functor cgf)

Add in the spec an implicit conversion of handler_event to queue& so it is possible to chain operations on the queue

Class cl::sycl::range< Dimensions >

use std::size_t Dimensions instead of int Dimensions in the specification?

add to the specification this default parameter value?

add to the specification some way to specify an offset?

Member cl::sycl::range< Dimensions >::get_count () const

Give back size() its real meaning in the specification

add this method to the specification

Namespace cl::sycl::trisycl

Refactor when updating to latest specification

Class cl::sycl::vec< DataType, NumElements >

add [] operator

add iterators on elements, with begin() and end()

having vec<> sub-classing array<> instead would solve the previous issues

move the implementation elsewhere

simplify the helpers by removing some template types since there are now inside the vec<> class.

rename in the specification element_type to value_type

Class cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >

Deal with multi-dimension array. Now, since we can only deal with 1-dim, PDim is set to 1 by default.

Class cl::sycl::xilinx::partition::complete< PDim >

Deal with multi-dimension array. Now, since we can only deal with 1-dim, PDim is set to 1 by default.

Class cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >

Deal with multi-dimension array. Now, since we can only deal with 1-dim, PDim is set to 1 by default.

Class cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >

Deal with multi-dimension array.

Member cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array (const Some↩ Container &src)

Find a way to specialize this with a safer implementation when the size of src is at least constexpr

Member cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array (std::initializer↩ _list< SourceBasicType > l)

Find a way to specialize this with a safer implementation when the size of src is at least constexpr This does not work... static_assert(l.size() == Size);

Member const_func

avoid this hack

Module execution

The implementation is quite minimal for now. :-)

Class handler_event

To be implemented

To be implemented

Member TRISYCL_ParallelForKernel_RANGE (N)

Add in the spec a version taking a kernel and a functor, to have host fall-back

Think to a cleaner solution

Think to a cleaner solution

Remove either task or q

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Data access and storage in SYCL 29
Dealing with OpenCL address spaces 205
Platforms, contexts, devices and queues 233
Helpers to do array and tuple conversion 338
Some helpers for the implementation 343
Debugging and tracing support 364
Manage default configuration and types 367
Device-side runtime implementation 369
Error handling 376
Expressing parallelism through kernels 403
Vector types in SYCL 464
Xilinx vendor extensions 470

Chapter 4

Namespace Index

4.1 Namespace List

Here is a list of all namespaces with brief descriptions:

cl
The vector type to be used as SYCL vector 479
cl::sycl 480
cl::sycl::access
Describe the type of access by kernels 488
cl::sycl::detail 490
cl::sycl::drt
The device-side runtime 496
cl::sycl::drt::code
The binary code of the kernels 496
cl::sycl::info 497
cl::sycl::trisycl 499
cl::sycl::xilinx 499
cl::sycl::xilinx::partition
Kind of array partition 501
std 502

Chapter 5

Hierarchical Index

5.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

cl::sycl::detail::accessor_base 29
cl::sycl::detail::accessor< T, Dimensions, Mode, Target > 29
cl::sycl::detail::address_space_base< T, AS > 205
cl::sycl::detail::address_space_object< T, AS > 205
cl::sycl::detail::address_space_variable< T, AS > 205
cl::sycl::detail::address_space_array< T, AS > 205
cl::sycl::detail::address_space_fundamental< T, AS > 205
cl::sycl::detail::address_space_ptr< T, AS > 205
array
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > 343
cl::sycl::detail::small_array< BasicType, FinalType, 1 > 343
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 > 343
cl::sycl::detail::small_array< BasicType, FinalType, 2 > 343
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 > 343
cl::sycl::detail::small_array< BasicType, FinalType, 3 > 343
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 > 343
cl::sycl::detail::small_array< BasicType, FinalType, Dims > 343
cl::sycl::detail::small_array_123< BasicType, FinalType, Dims > 343
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > 343
cl::sycl::vec< DataType, NumElements > 464
cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > 343
cl::sycl::detail::small_array_123< std::size_t, id< Dimensions >, Dimensions > 343
cl::sycl::id< Dimensions > 403
cl::sycl::id< dimensionality > 403
cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > 343
cl::sycl::detail::small_array_123< std::size_t, range< Dimensions >, Dimensions > 343
cl::sycl::range< Dimensions > 403
cl::sycl::range< dimensionality > 403
bitwise
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > 343
cl::sycl::detail::small_array< BasicType, FinalType, 1 > 343
cl::sycl::detail::small_array< BasicType, FinalType, 2 > 343
cl::sycl::detail::small_array< BasicType, FinalType, 3 > 343

cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 343
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 343
cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . 343
cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . 343
cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim > . . . 503
cl::sycl::detail::cache< Key, Value > . . . 505
cl::sycl::detail::cache< cl_command_queue, cl::sycl::detail::opencl_queue > . . . 505
cl::sycl::detail::cache< cl_context, cl::sycl::detail::opencl_context > . . . 505
cl::sycl::detail::cache< cl_device_id, cl::sycl::detail::opencl_device > . . . 505
cl::sycl::detail::cache< cl_kernel, cl::sycl::detail::opencl_kernel > . . . 505
cl::sycl::detail::cache< cl_platform_id, cl::sycl::detail::opencl_platform > . . . 505
cl::sycl::xilinx::partition::complete< PDim > . . . 510
cl::sycl::detail::container_element_aspect< T > . . . 343
cl::sycl::detail::container_element_aspect< Accessor::value_type > . . . 343
cl::sycl::drt::accessor< Accessor > . . . 369
cl::sycl::detail::container_element_aspect< DataType > . . . 343
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target > . . . 29
cl::sycl::detail::context . . . 233
cl::sycl::detail::host_context . . . 233
cl::sycl::detail::opencl_context . . . 533
cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim > . . . 512
cl::sycl::detail::debug< T > . . . 364
cl::sycl::detail::debug< accessor< T, Dimensions, Mode, access::target::local > > . . . 364
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local > . . . 29
cl::sycl::detail::debug< accessor< T, Dimensions, Mode, Target > > . . . 364
cl::sycl::detail::accessor< T, Dimensions, Mode, Target > . . . 29
cl::sycl::detail::debug< buffer< T, Dimensions > > . . . 364
cl::sycl::detail::buffer< T, Dimensions > . . . 29
cl::sycl::detail::debug< buffer< T, Dimensions, Allocator > > . . . 364
cl::sycl::buffer< T, Dimensions, Allocator > . . . 29
cl::sycl::detail::debug< buffer_waiter< T, Dimensions, Allocator > > . . . 364
cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator > . . . 29
cl::sycl::detail::debug< detail::kernel > . . . 364
cl::sycl::detail::kernel . . . 233
cl::sycl::detail::opencl_kernel . . . 548
cl::sycl::detail::debug< detail::pipe_accessor< DataType, AccessMode, Target > > . . . 364
cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe > . . . 29
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe > . . . 29
cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::pipe > . . . 29
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe > . . . 29
cl::sycl::detail::debug< detail::pipe_accessor< T, AccessMode, Target > > . . . 364
cl::sycl::detail::pipe_accessor< T, AccessMode, Target > . . . 29
cl::sycl::detail::debug< detail::pipe_reservation< PipeAccessor > > . . . 364
cl::sycl::detail::pipe_reservation< PipeAccessor > . . . 29
cl::sycl::detail::debug< detail::queue > . . . 364
cl::sycl::detail::queue . . . 563
cl::sycl::detail::host_queue . . . 528
cl::sycl::detail::opencl_queue . . . 554
cl::sycl::detail::debug< handler > . . . 364
cl::sycl::handler . . . 233
cl::sycl::detail::debug< host_queue > . . . 364
cl::sycl::detail::host_queue . . . 528
cl::sycl::detail::debug< opencl_kernel > . . . 364

cl::sycl::detail::opencl_kernel . . . 548
cl::sycl::detail::debug< opencl_queue > . . . 364
cl::sycl::detail::opencl_queue . . . 554
cl::sycl::detail::debug< pipe< T > > . . . 364
cl::sycl::detail::pipe< T > . . . 29
cl::sycl::pipe< T > . . . 29
cl::sycl::detail::debug< pipe< value_type > > . . . 364
cl::sycl::detail::pipe< value_type > . . . 29
cl::sycl::detail::debug< queue > . . . 364
cl::sycl::queue . . . 233
cl::sycl::detail::debug< static_pipe< T, Capacity > > . . . 364
cl::sycl::static_pipe< T, Capacity > . . . 29
cl::sycl::detail::debug< task > . . . 364
cl::sycl::detail::task . . . 576
cl::sycl::detail::device . . . 233
cl::sycl::detail::host_device . . . 523
cl::sycl::detail::opencl_device . . . 541
cl::sycl::device_selector . . . 233
cl::sycl::device_type_selector . . . 233
cl::sycl::device_typename_selector< DeviceType > . . . 233
cl::sycl::detail::display_vector< T > . . . 364
cl::sycl::detail::display_vector< FinalType > . . . 364
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 343
cl::sycl::detail::display_vector< id< Dimensions > > . . . 364
cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . 343
cl::sycl::detail::display_vector< range< Dimensions > > . . . 364
cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . 343
cl::sycl::detail::display_vector< vec< DataType, NumElements > > . . . 364
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 343
enable_shared_from_this
cl::sycl::detail::accessor< T, Dimensions, Mode, Target > . . . 29
cl::sycl::detail::buffer_base . . . 29
cl::sycl::detail::buffer< T, Dimensions > . . . 29
cl::sycl::detail::task . . . 576
cl::sycl::error_handler . . . 376
cl::sycl::trisycl::default_error_handler . . . 514
euclidean_ring_operators
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 343
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 343
cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . 343
cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . 343
cl::sycl::event . . . 515
cl::sycl::exception . . . 376
cl::sycl::async_exception . . . 376
cl::sycl::cl_exception . . . 376
cl::sycl::device_error . . . 376

cl::sycl::compile_program_error . . . 376
cl::sycl::feature_not_supported . . . 376
cl::sycl::invalid_object_error . . . 376
cl::sycl::link_program_error . . . 376
cl::sycl::memory_allocation_error . . . 376
cl::sycl::platform_error . . . 376
cl::sycl::profiling_error . . . 376
cl::sycl::runtime_error . . . 376
cl::sycl::accessor_error . . . 376
cl::sycl::event_error . . . 376
cl::sycl::invalid_parameter_error . . . 376
cl::sycl::kernel_error . . . 376
cl::sycl::nd_range_error . . . 376
cl::sycl::non_cl_error . . . 376
cl::sycl::pipe_error . . . 376
cl::sycl::detail::expand_to_vector< V, Tuple, expansion > . . . 338
cl::sycl::detail::expand_to_vector< V, Tuple, true > . . . 338
false_type
cl::sycl::is_wrapper< T > . . . 531
cl::sycl::group< Dimensions > . . . 403
handler_event . . . 516
std::hash< cl::sycl::buffer< T, Dimensions, Allocator > > . . . 516
std::hash< cl::sycl::context > . . . 517
std::hash< cl::sycl::device > . . . 518
std::hash< cl::sycl::kernel > . . . 519
std::hash< cl::sycl::platform > . . . 520
std::hash< cl::sycl::queue > . . . 521
cl::sycl::image< Dimensions > . . . 29
cl::sycl::item< Dimensions > . . . 403
cl::sycl::nd_item< Dimensions > . . . 403
cl::sycl::nd_range< Dimensions > . . . 403
cl::sycl::xilinx::partition::none . . . 532
cl::sycl::detail::ocl_type< T, AS > . . . 205
cl::sycl::detail::ocl_type< T, constant_address_space > . . . 205
cl::sycl::detail::ocl_type< T, global_address_space > . . . 205
cl::sycl::detail::ocl_type< T, local_address_space > . . . 205
cl::sycl::detail::ocl_type< T, private_address_space > . . . 205
cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id > . . . 403
cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id > . . . 403
cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id > . . . 403
cl::sycl::info::param_traits< T, Param > . . . 560
cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType > . . . 470
cl::sycl::pipe_reservation< PipeAccessor > . . . 29
cl::sycl::detail::platform . . . 233
cl::sycl::detail::host_platform . . . 233
cl::sycl::detail::opencl_platform . . . 233
cl::sycl::drt::code::program . . . 561
cl::sycl::detail::reserve_id< T > . . . 29
shiftable
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 1 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 2 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, 3 > . . . 343
cl::sycl::detail::small_array< BasicType, FinalType, Dims > . . . 343
cl::sycl::detail::small_array< DataType, vec< DataType, NumElements >, NumElements > . . . 343
cl::sycl::detail::small_array< std::size_t, id< Dimensions >, Dims > . . . 343
cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims > . . . 343
cl::sycl::detail::singleton< T > . . . 575

cl::sycl::detail::singleton< host_context > . . . 575
cl::sycl::detail::host_context . . . 233
cl::sycl::detail::singleton< host_device > . . . 575
cl::sycl::detail::host_device . . . 523
cl::sycl::detail::singleton< host_platform > . . . 575
cl::sycl::detail::host_platform . . . 233
totally_ordered
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation > . . . 570
cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > > . . . 570
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target > . . . 29
cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > > . . . 570
cl::sycl::buffer< T, Dimensions, Allocator > . . . 29
cl::sycl::detail::shared_ptr_implementation< buffer_waiter< T, Dimensions, Allocator >, detail::buffer< T, Dimensions > > . . . 570
cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator > . . . 29
cl::sycl::detail::shared_ptr_implementation< context, detail::context > . . . 570
cl::sycl::context . . . 233
cl::sycl::detail::shared_ptr_implementation< device, detail::device > . . . 570
cl::sycl::device . . . 233
cl::sycl::detail::shared_ptr_implementation< kernel, detail::kernel > . . . 570
cl::sycl::kernel . . . 233
cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > > . . . 570
cl::sycl::pipe< T > . . . 29
cl::sycl::detail::shared_ptr_implementation< platform, detail::platform > . . . 570
cl::sycl::platform . . . 233
cl::sycl::detail::shared_ptr_implementation< queue, detail::queue > . . . 570
cl::sycl::queue . . . 233
cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > > . . . 570
cl::sycl::static_pipe< T, Capacity > . . . 29
type
cl::sycl::detail::address_space_object< T, AS > . . . 205
vector
cl::sycl::exception_list . . . 376
bool . . . ??
shared_ptr< cl::sycl::detail::pipe< DataType > > . . . ??
shared_ptr< detail::accessor< DataType, Dimensions, AccessMode, Target > > . . . ??
shared_ptr< detail::buffer< T, Dimensions > > . . . ??
shared_ptr< detail::buffer_waiter< T, Dimensions, Allocator > > . . . ??
shared_ptr< detail::context > . . . ??
shared_ptr< detail::device > . . . ??
shared_ptr< detail::kernel > . . . ??
shared_ptr< detail::pipe< T > > . . . ??
shared_ptr< detail::platform > . . . ??
shared_ptr< detail::queue > . . . ??
static const size_t . . . ??
static constexpr bool . . . ??

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >
Represent a block partition . . . 503
cl::sycl::detail::cache< Key, Value >
A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys . . 505
cl::sycl::xilinx::partition::complete< PDim >
Represent a complete partition . . . 510
cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >
Represent a cyclic partition . . . 512
cl::sycl::trisycl::default_error_handler . . . 514
cl::sycl::event . . . 515
handler_event
Handler event . . . 516
std::hash< cl::sycl::buffer< T, Dimensions, Allocator > > . . . 516
std::hash< cl::sycl::context > . . . 517
std::hash< cl::sycl::device > . . . 518
std::hash< cl::sycl::kernel > . . . 519
std::hash< cl::sycl::platform > . . . 520
std::hash< cl::sycl::queue > . . . 521
cl::sycl::detail::host_device
SYCL host device . . . 523
cl::sycl::detail::host_queue
Some implementation details about the SYCL queue . . . 528
cl::sycl::is_wrapper< T > . . . 531
cl::sycl::xilinx::partition::none
Represent a none partition . . . 532
cl::sycl::detail::opencl_context
SYCL OpenCL context . . . 533
cl::sycl::detail::opencl_device
SYCL OpenCL device . . . 541
cl::sycl::detail::opencl_kernel
An abstraction of the OpenCL kernel . . . 548
cl::sycl::detail::opencl_queue
Some implementation details about the SYCL queue . . . 554
cl::sycl::info::param_traits< T, Param >
Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value) . . . 560

cl::sycl::drt::code::program
A binary program containing some kernels 561
cl::sycl::detail::queue
Some implementation details about the SYCL queue 563
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >
Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers 570
cl::sycl::detail::singleton< T >
Provide a singleton factory 575
cl::sycl::detail::task
The abstraction to represent SYCL tasks executing inside command_group 576

Chapter 7

File Index

7.1 File List

Here is a list of all files with brief descriptions:

include/CL/sycl.hpp . . . 593
include/CL/sycl/access.hpp . . . 595
include/CL/sycl/accessor.hpp . . . 597
include/CL/sycl/address_space.hpp
Implement OpenCL address spaces in SYCL with C++-style . . . 627
include/CL/sycl/allocator.hpp . . . 631
include/CL/sycl/buffer.hpp . . . 640
include/CL/sycl/buffer_allocator.hpp . . . 656
include/CL/sycl/context.hpp . . . 665
include/CL/sycl/device.hpp . . . 987
include/CL/sycl/device_runtime.hpp . . . 1004
include/CL/sycl/device_selector.hpp . . . 1009
include/CL/sycl/error_handler.hpp . . . 1015
include/CL/sycl/event.hpp . . . 1017
include/CL/sycl/exception.hpp . . . 1018
include/CL/sycl/group.hpp . . . 1023
include/CL/sycl/handler.hpp . . . 1027
include/CL/sycl/handler_event.hpp . . . 1037
include/CL/sycl/id.hpp . . . 1038
include/CL/sycl/image.hpp
OpenCL SYCL image class . . . 1040
include/CL/sycl/item.hpp . . . 1053
include/CL/sycl/kernel.hpp . . . 1059
include/CL/sycl/math.hpp
Implement a wrapper around OpenCL math operations Joan.Thibault AT ens-rennes POINT fr
This file is distributed under the University of Illinois Open Source License . . . 1066
include/CL/sycl/nd_item.hpp . . . 1072
include/CL/sycl/nd_range.hpp . . . 1077
include/CL/sycl/opencl_types.hpp
TriSYCL wrapper for OpenCL types . . . 1079
include/CL/sycl/parallelism.hpp
Implement parallel constructions to launch kernels . . . 1094
include/CL/sycl/pipe.hpp . . . 1103
include/CL/sycl/pipe_reservation.hpp . . . 1114
include/CL/sycl/platform.hpp . . . 1050

include/CL/sycl/queue.hpp . . . 1134
include/CL/sycl/range.hpp . . . 1140
include/CL/sycl/static_pipe.hpp . . . 1143
include/CL/sycl/vec.hpp
Implement the small OpenCL vector class . . . 1145
include/CL/sycl/accessor/detail/accessor_base.hpp . . . 611
include/CL/sycl/accessor/detail/local_accessor.hpp . . . 613
include/CL/sycl/address_space/detail/address_space.hpp
Implement OpenCL address spaces in SYCL with C++-style . . . 619
include/CL/sycl/buffer/detail/accessor.hpp . . . 604
include/CL/sycl/buffer/detail/buffer.hpp . . . 633
include/CL/sycl/buffer/detail/buffer_base.hpp . . . 648
include/CL/sycl/buffer/detail/buffer_waiter.hpp . . . 653
include/CL/sycl/command_group/detail/task.hpp . . . 658
include/CL/sycl/context/detail/context.hpp . . . 663
include/CL/sycl/context/detail/host_context.hpp . . . 671
include/CL/sycl/context/detail/opencl_context.hpp . . . 674
include/CL/sycl/detail/array_tuple_helpers.hpp
Some helpers to do array-tuple conversions . . . 677
include/CL/sycl/detail/cache.hpp . . . 681
include/CL/sycl/detail/container_element_aspect.hpp . . . 684
include/CL/sycl/detail/debug.hpp . . . 685
include/CL/sycl/detail/default_classes.hpp . . . 690
include/CL/sycl/detail/global_config.hpp . . . 693
include/CL/sycl/detail/instantiate_kernel.hpp . . . 697
include/CL/sycl/detail/linear_id.hpp . . . 700
include/CL/sycl/detail/shared_ptr_implementation.hpp . . . 701
include/CL/sycl/detail/singleton.hpp . . . 704
include/CL/sycl/detail/small_array.hpp . . . 706
include/CL/sycl/detail/unimplemented.hpp . . . 982
include/CL/sycl/detail/SPIR/opencl_spir.h . . . 711
include/CL/sycl/device/detail/device.hpp . . . 984
include/CL/sycl/device/detail/device_tail.hpp . . . 997
include/CL/sycl/device/detail/host_device.hpp . . . 998
include/CL/sycl/device/detail/opencl_device.hpp . . . 1001
include/CL/sycl/device_selector/detail/device_selector_tail.hpp . . . 1011
include/CL/sycl/info/context.hpp . . . 669
include/CL/sycl/info/device.hpp . . . 992
include/CL/sycl/info/param_traits.hpp . . . 1042
include/CL/sycl/info/platform.hpp . . . 1044
include/CL/sycl/kernel/detail/kernel.hpp . . . 1056
include/CL/sycl/kernel/detail/opencl_kernel.hpp . . . 1062
include/CL/sycl/parallelism/detail/parallelism.hpp
Implement the detail of the parallel constructions to launch kernels . . . 1087
include/CL/sycl/pipe/detail/pipe.hpp . . . 1096
include/CL/sycl/pipe/detail/pipe_accessor.hpp . . . 1106
include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp . . . 1110
include/CL/sycl/platform/detail/host_platform.hpp . . . 1117
include/CL/sycl/platform/detail/host_platform_tail.hpp . . . 1121
include/CL/sycl/platform/detail/opencl_platform.hpp . . . 1122
include/CL/sycl/platform/detail/opencl_platform_tail.hpp . . . 1125
include/CL/sycl/platform/detail/platform.hpp . . . 1047
include/CL/sycl/queue/detail/host_queue.hpp . . . 1126
include/CL/sycl/queue/detail/opencl_queue.hpp . . . 1129
include/CL/sycl/queue/detail/queue.hpp . . . 1131
include/CL/sycl/vendor/Xilinx/opt_decorate_func.hpp . . . 1150
include/CL/sycl/vendor/Xilinx/partition_array.hpp . . . 1151
include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp . . . 1156

Chapter 8

Module Documentation

8.1 Data access and storage in SYCL

Namespaces

- cl::sycl::access

  Describe the type of access by kernels.

Classes

- class cl::sycl::detail::accessor_base

  The buffer accessor base is here mainly to be accessed from the task without dependency on the buffer data types. More...
- class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >

  The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group. More...
- class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

  The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::detail::buffer< T, Dimensions >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- struct cl::sycl::detail::buffer_base

  Factorize some template independent buffer aspects in a base class. More...
- class cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >

  A helper class to wait for the final buffer destruction if the conditions for blocking are met. More...
- class cl::sycl::buffer< T, Dimensions, Allocator >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...

- struct cl::sycl::image< Dimensions >
- struct cl::sycl::detail::reserve_id< T >

  A private description of a reservation station. More...
- class cl::sycl::detail::pipe< T >

  Implement a pipe object. More...
- class cl::sycl::detail::pipe_accessor< T, AccessMode, Target >

  The accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::pipe< T >

  A SYCL pipe. More...
- class cl::sycl::detail::pipe_reservation< PipeAccessor >

  The implementation of the pipe reservation station. More...
- struct cl::sycl::pipe_reservation< PipeAccessor >

  The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example. More...
- class cl::sycl::static_pipe< T, Capacity >

  A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. More...

Typedefs

- template<typename T >
  using cl::sycl::buffer_allocator = std::allocator< T >

  The allocator objects give the programmer some control on how the memory is allocated inside SYCL.
- template<typename T >
  using cl::sycl::image_allocator = std::allocator< T >

  The allocator used for the `image` inside SYCL.
- template<typename T >
  using cl::sycl::map_allocator = std::allocator< T >

  The allocator used to map the memory at the same place.

Functions

- template<typename Accessor >
  static auto & cl::sycl::get_pipe_detail (Accessor &a)

  Top-level function to break circular dependencies on the the types to get the pipe implementation.
- template<typename BufferDetail >
  static std::shared_ptr< detail::task > cl::sycl::detail::buffer_add_to_task (BufferDetail buf, handler *command_group_handler, bool is_write_mode)

  Proxy function to avoid some circular type recursion.
- static std::shared_ptr< detail::task > cl::sycl::detail::add_buffer_to_task (handler *command_group_handler, std::shared_ptr< detail::buffer_base > b, bool is_write_mode)
- template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>
  auto cl::sycl::detail::waiter (detail::buffer< T, Dimensions > *b)

  Helper function to create a new buffer_waiter.

8.1.1 Detailed Description

8.1.2 Class Documentation

8.1.2.1 class cl::sycl::detail::accessor_base

The buffer accessor base is here mainly to be accessed from the task without dependency on the buffer data types.

Definition at line 39 of file accessor_base.hpp.

Inheritance diagram for cl::sycl::detail::accessor_base:

Collaboration diagram for cl::sycl::detail::accessor_base:

Public Member Functions

- void set_order (std::size_t o)

  Set the order of the accessor.
- auto get_order ()

  Get the order of the accessor.
- virtual boost::compute::buffer get_cl_buffer () const =0

  Get the boost::compute::buffer or throw if unset.

Protected Attributes

- std::shared_ptr< detail::task > task

  The task where the accessor is used in.
- std::size_t order

  The declaration order of the accessor.

8.1.2.1.1 Member Function Documentation

8.1.2.1.1.1 get_cl_buffer()

```
virtual boost::compute::buffer cl::sycl::detail::accessor_base::get_cl_buffer ( ) const [pure virtual]
```

Get the boost::compute::buffer or throw if unset.

Implemented in cl::sycl::detail::accessor< T, Dimensions, Mode, Target >.

Referenced by get_order().

Here is the caller graph for this function:

8.1.2.1.1.2 get_order()

```
auto cl::sycl::detail::accessor_base::get_order ( ) [inline]
```

Get the order of the accessor.

Definition at line 58 of file accessor_base.hpp.

References get_cl_buffer(), and order.

```
00058                    {
00059     return order;
00060   }
```

Here is the call graph for this function:

8.1.2.1.1.3 set_order()

```
void cl::sycl::detail::accessor_base::set_order (
            std::size_t o )  [inline]
```

Set the order of the accessor.

Definition at line 52 of file accessor_base.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor().

```
00052                                {
00053     order = o;
00054   }
```

Here is the caller graph for this function:

8.1.2.1.2 Member Data Documentation

8.1.2.1.2.1 order

```
std::size_t cl::sycl::detail::accessor_base::order  [protected]
```

The declaration order of the accessor.

Definition at line 47 of file accessor_base.hpp.

Referenced by get_order().

8.1.2.1.2.2 task

```
std::shared_ptr<detail::task> cl::sycl::detail::accessor_base::task  [protected]
```

The task where the accessor is used in.

Definition at line 44 of file accessor_base.hpp.

8.1.2.2 class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >

template<typename T, int Dimensions, access::mode Mode>
class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >

The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group.

Todo Use the access::mode

Definition at line 54 of file local_accessor.hpp.

Inheritance diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >:

Collaboration diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >:

Public Types

- using value_type = T
- using element = T
- using reference = typename array_type::reference
- using const_reference = typename array_type::const_reference
- using iterator = typename array_type::iterator

  Inherit the iterator types from the implementation.
- using const_iterator = typename array_type::const_iterator
- using reverse_iterator = typename array_type::reverse_iterator
- using const_reverse_iterator = typename array_type::const_reverse_iterator

Public Member Functions

- accessor (const range< Dimensions > &allocation_size, handler &command_group_handler)

  Construct a device accessor from an existing buffer.
- ~accessor ()
- auto get_range () const

  Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.
- auto get_count () const

  Returns the total number of elements behind the accessor.
- auto get_size () const

  Returns the size of the underlying buffer storage in bytes.
- reference operator[ ] (std::size_t index)

  Use the accessor with integers à la [][][].
- reference operator[ ] (std::size_t index) const

  Use the accessor with integers à la [][][].
- auto & operator[ ] (id< dimensionality > index)

  To use the accessor with [id<>].
- auto & operator[ ] (id< dimensionality > index) const

  To use the accessor with [id<>].
- auto & operator[ ] (item< dimensionality > index)

  To use an accessor with [item<>].
- auto & operator[ ] (item< dimensionality > index) const

  To use an accessor with [item<>].
- auto & operator[ ] (nd_item< dimensionality > index)

  To use an accessor with an [nd_item<>].
- auto & operator[ ] (nd_item< dimensionality > index) const

  To use an accessor with an [nd_item<>].
- reference operator* ()

  Get the first element of the accessor.
- reference operator* () const

  Get the first element of the accessor.
- constexpr bool is_read_access () const

  Test if the accessor has a read access right.
- constexpr bool is_write_access () const

  Test if the accessor has a write access right.
- iterator begin () const

  Forward all the iterator functions to the implementation.
- iterator end () const
- const_iterator cbegin () const
- const_iterator cend () const
- reverse_iterator rbegin () const
- reverse_iterator rend () const
- const_reverse_iterator crbegin () const
- const_reverse_iterator crend () const

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Types

- using array_type = boost::multi_array_ref< T, Dimensions >

  The implementation is a multi_array_ref wrapper.
- using writable_array_type = typename std::remove_const< array_type >::type

Private Member Functions

- auto allocate_accessor (const range< Dimensions > &r)

  Allocate uninitialized buffer memory.
- void deallocate_accessor ()

  Deallocate accessor memory.

Private Attributes

- writable_array_type array

  The way the buffer is really accessed.
- T * allocation = nullptr

  The allocation on the host for the local accessor.
- friend handler

8.1.2.2.1 Member Typedef Documentation

8.1.2.2.1.1 array_type

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::array_type =
boost::multi_array_ref<T, Dimensions>  [private]
```

The implementation is a multi_array_ref wrapper.

Definition at line 61 of file local_accessor.hpp.

8.1.2.2.1.2 const_iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::const_iterator
= typename array_type::const_iterator
```

Definition at line 104 of file local_accessor.hpp.

8.1.2.2.1.3 const_reference

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::const_reference
= typename array_type::const_reference
```

Definition at line 97 of file local_accessor.hpp.

8.1.2.2.1.4 const_reverse_iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::const_reverse↩
_iterator = typename array_type::const_reverse_iterator
```

Definition at line 107 of file local_accessor.hpp.

8.1.2.2.1.5 element

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::element = T
```

Definition at line 95 of file local_accessor.hpp.

8.1.2.2.1.6 iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::iterator =
typename array_type::iterator
```

Inherit the iterator types from the implementation.

Todo Add iterators to accessors in the specification

Definition at line 103 of file local_accessor.hpp.

8.1.2.2.1.7 reference

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::reference =
typename array_type::reference
```

Definition at line 96 of file local_accessor.hpp.

8.1.2.2.1.8 reverse_iterator

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::reverse_↩
iterator = typename array_type::reverse_iterator
```

Definition at line 105 of file local_accessor.hpp.

8.1.2.2.1.9 value_type

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::value_type = T
```

Todo in the specification: store the types for user request as STL or C++AMP

Definition at line 94 of file local_accessor.hpp.

8.1.2.2.1.10 writable_array_type

```
template<typename T , int Dimensions, access::mode Mode>
using cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::writable_↩
array_type = typename std::remove_const<array_type>::type  [private]
```

Definition at line 66 of file local_accessor.hpp.

8.1.2.2.2 Constructor & Destructor Documentation

8.1.2.2.2.1 accessor()

```
template<typename T , int Dimensions, access::mode Mode>
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::accessor (
            const range< Dimensions > & allocation_size,
            handler & command_group_handler )  [inline]
```

Construct a device accessor from an existing buffer.

Todo fix the specification to rename target that shadows template param

Definition at line 115 of file local_accessor.hpp.

```
00116                                          :
00117     array { allocate_accessor(allocation_size) } {}
```

8.1.2.2.2.2 ~accessor()

```
template<typename T , int Dimensions, access::mode Mode>
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::~accessor ( )  [inline]
```

Definition at line 121 of file local_accessor.hpp.

```
00121                {
00122      deallocate_accessor();
00123    }
```

8.1.2.2.3 Member Function Documentation

8.1.2.2.3.1 allocate_accessor()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::allocate_↩
accessor (
            const range< Dimensions > & r )  [inline], [private]
```

Allocate uninitialized buffer memory.

Definition at line 364 of file local_accessor.hpp.

References cl::sycl::range< Dimensions >::get_count().

```
00364                                                    {
00365      auto count = r.get_count();
00366      // Allocate uninitialized memory
00367      allocation = std::allocator<value_type>{}.allocate(count);
00368      return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
00369    }
```

Here is the call graph for this function:

8.1.2.2.3.2 begin()

```
template<typename T , int Dimensions, access::mode Mode>
iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::begin ( )
const  [inline]
```

Forward all the iterator functions to the implementation.

Todo Add these functions to the specification

Todo The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

Todo try to solve it by using some enable_if on array constness?

Todo The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Todo Factor out these in a template helper

Todo Do we need this in detail::accessor too or only in accessor?

Definition at line 315 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array, and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin().

```
00315                                {
00316     return const_cast<writable_array_type &>(array).
      begin();
00317   }
```

Here is the call graph for this function:

8.1.2.2.3.3 cbegin()

```
template<typename T , int Dimensions, access::mode Mode>
const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↩
::cbegin ( ) const  [inline]
```

Definition at line 332 of file local_accessor.hpp.

```
00332 { return array.begin(); }
```

8.1.2.2.3.4 cend()

```
template<typename T , int Dimensions, access::mode Mode>
const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::cend
( ) const  [inline]
```

Definition at line 335 of file local_accessor.hpp.

```
00335 { return array.end(); }
```

8.1.2.2.3.5 crbegin()

```
template<typename T , int Dimensions, access::mode Mode>
const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local
>::crbegin ( ) const  [inline]
```

Definition at line 356 of file local_accessor.hpp.

```
00356 { return array.rbegin(); }
```

8.1.2.2.3.6 crend()

```
template<typename T , int Dimensions, access::mode Mode>
const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local
>::crend ( ) const  [inline]
```

Definition at line 359 of file local_accessor.hpp.

```
00359 { return array.rend(); }
```

8.1.2.2.3.7 deallocate_accessor()

```
template<typename T , int Dimensions, access::mode Mode>
void cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::deallocate_↩
accessor ( ) [inline], [private]
```

Deallocate accessor memory.

Definition at line 373 of file local_accessor.hpp.

```
00373                                {
00374     std::allocator<value_type>{}.deallocate(allocation, array.num_elements());
00375   }
```

8.1.2.2.3.8 end()

```
template<typename T , int Dimensions, access::mode Mode>
iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::end ( )
const [inline]
```

Definition at line 321 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array, and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::end().

```
00321                          {
00322     return const_cast<writable_array_type &>(array).end();
00323   }
```

Here is the call graph for this function:

8.1.2.2.3.9 get_count()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_count ( )
const [inline]
```

Returns the total number of elements behind the accessor.

Equal to get_range()[0] ∗ ... ∗ get_range()[Dimensions-1].

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 156 of file local_accessor.hpp.

```
00156                          {
00157     return array.num_elements();
00158   }
```

8.1.2.2.3.10 get_range()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_range ( )
const [inline]
```

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 134 of file local_accessor.hpp.

```
00134                          {
00135     /* Interpret the shape which is a pointer to the first element as an
00136        array of Dimensions elements so that the range<Dimensions>
00137        constructor is happy with this collection
00138
00139        \todo Add also a constructor in range<> to accept a const
00140        std::size_t *?
00141     */
00142     return range<Dimensions> {
00143       *(const std::size_t (*)[Dimensions])(array.shape())
00144         };
00145   }
```

8.1.2.2.3.11 get_size()

```
template<typename T , int Dimensions, access::mode Mode>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_size ( )
const [inline]
```

Returns the size of the underlying buffer storage in bytes.

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 167 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count().

```
00167                          {
00168     return get_count()*sizeof(value_type);
00169   }
```

Here is the call graph for this function:

8.1.2.2.3.12 is_read_access()

```
template<typename T , int Dimensions, access::mode Mode>
constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::is_↩
read_access ( ) const [inline]
```

Test if the accessor has a read access right.

Todo Strangely, it is not really constexpr because it is not a static method...

Todo to move in the access::mode enum class and add to the specification ?

Definition at line 267 of file local_accessor.hpp.

References cl::sycl::access::discard_read_write, cl::sycl::access::read, and cl::sycl::access::read_write.

```
00267                                       {
00268     return Mode == access::mode::read
00269       || Mode == access::mode::read_write
00270       || Mode == access::mode::discard_read_write;
00271   }
```

8.1.2.2.3.13 is_write_access()

```
template<typename T , int Dimensions, access::mode Mode>
constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::is_↩
write_access ( ) const [inline]
```

Test if the accessor has a write access right.

Todo Strangely, it is not really constexpr because it is not a static method...

Todo to move in the access::mode enum class and add to the specification ?

Definition at line 282 of file local_accessor.hpp.

References cl::sycl::access::discard_read_write, cl::sycl::access::discard_write, cl::sycl::access::read_write, and cl::sycl::access::write.

```
00282                                       {
00283     return Mode == access::mode::write
00284       || Mode == access::mode::read_write
00285       || Mode == access::mode::discard_write
00286       || Mode == access::mode::discard_read_write;
00287   }
```

8.1.2.2.3.14 operator*() [1/2]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator*
( )  [inline]
```

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification

Definition at line 239 of file local_accessor.hpp.

```
00239                          {
00240     return *array.data();
00241   }
```

8.1.2.2.3.15 operator*() [2/2]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator*
( ) const  [inline]
```

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification?

Todo Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Definition at line 254 of file local_accessor.hpp.

```
00254                                {
00255     return *array.data();
00256   }
```

8.1.2.2.3.16 operator[]() [1/8]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[]
(
            std::size_t index )  [inline]
```

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 177 of file local_accessor.hpp.

```
00177                                        {
00178     return array[index];
00179   }
```

8.1.2.2.3.17 operator[]() [2/8]

```
template<typename T , int Dimensions, access::mode Mode>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[]
(
            std::size_t index ) const  [inline]
```

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 187 of file local_accessor.hpp.

```
00187                                                {
00188     return array[index];
00189   }
```

8.1.2.2.3.18 operator[]() [3/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
            id< dimensionality > index )  [inline]
```

To use the accessor with [id<>].

Definition at line 193 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
00193                                                {
00194     return array(index);
00195   }
```

8.1.2.2.3.19 operator[]() [4/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
            id< dimensionality > index ) const  [inline]
```

To use the accessor with [id<>].

Definition at line 199 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
00199                                                {
00200     return array(index);
00201   }
```

8.1.2.2.3.20 operator[]() [5/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
            item< dimensionality > index )  [inline]
```

To use an accessor with [item<>].

Definition at line 205 of file local_accessor.hpp.

```
00205                                                {
00206     return (*this)[index.get()];
00207   }
```

8.1.2.2.3.21 operator[]() [6/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
            item< dimensionality > index ) const  [inline]
```

To use an accessor with [item<>].

Definition at line 211 of file local_accessor.hpp.

```
00211                                                      {
00212     return (*this)[index.get()];
00213   }
```

8.1.2.2.3.22 operator[]() [7/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
            nd_item< dimensionality > index )  [inline]
```

To use an accessor with an [nd_item<>].

Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 220 of file local_accessor.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

```
00220                                                    {
00221     return (*this)[index.get_global()];
00222   }
```

Here is the call graph for this function:

8.1.2.2.3.23 operator[]() [8/8]

```
template<typename T , int Dimensions, access::mode Mode>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[] (
            nd_item< dimensionality > index ) const  [inline]
```

To use an accessor with an [nd_item<>].

Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 228 of file local_accessor.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

```
00228                                                    {
00229     return (*this)[index.get_global()];
00230   }
```

Here is the call graph for this function:

8.1.2.2.3.24 rbegin()

```
template<typename T , int Dimensions, access::mode Mode>
reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↩
::rbegin ( ) const  [inline]
```

Definition at line 339 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array, and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rbegin().

```
00339                                    {
00340     return const_cast<writable_array_type &>(array).
      rbegin();
00341   }
```

Here is the call graph for this function:

8.1.2.2.3.25 rend()

```
template<typename T , int Dimensions, access::mode Mode>
reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↩
::rend ( ) const  [inline]
```

Definition at line 345 of file local_accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array, and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rend().

```
00345                                {
00346     return const_cast<writable_array_type &>(array).rend();
00347   }
```

Here is the call graph for this function:

8.1.2.2.4 Member Data Documentation

8.1.2.2.4.1 allocation

```
template<typename T , int Dimensions, access::mode Mode>
T* cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::allocation =
nullptr  [mutable], [private]
```

The allocation on the host for the local accessor.

Note that this is uninitialized memory, as stated in SYCL specification.

Definition at line 82 of file local_accessor.hpp.

8.1.2.2.4.2 array

```
template<typename T , int Dimensions, access::mode Mode>
writable_array_type cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↩
::array  [mutable], [private]
```

The way the buffer is really accessed.

Use a mutable member because the accessor needs to be captured by value in the lambda which is then read-only. This is to avoid the user to use mutable lambda or have a lot of const_cast as previously done in this implementation

Definition at line 75 of file local_accessor.hpp.

8.1.2.2.4.3 dimensionality

```
template<typename T , int Dimensions, access::mode Mode>
constexpr auto cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >↩
::dimensionality = Dimensions  [static]
```

Todo in the specification: store the dimension for user request

Todo Use another name, such as from C++17 committee discussions.

Definition at line 90 of file local_accessor.hpp.

8.1.2.2.4.4 handler

```
template<typename T , int Dimensions, access::mode Mode>
friend cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::handler
[private]
```

Definition at line 379 of file local_accessor.hpp.

8.1.2.3 class cl::sycl::accessor

template<typename DataType, int Dimensions, access::mode AccessMode, access::target Target = access::target::global_↩ buffer>
class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way.

Todo Implement it for images according so section 3.3.4.5

Definition at line 47 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >:

Collaboration diagram for cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >:

Public Member Functions

- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler)

  Construct a buffer accessor from a buffer using a command group handler object from the command group scope.
- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer)

  Construct a buffer accessor from a buffer.
- template<typename Allocator >
  accessor (buffer< DataType, Dimensions, Allocator > &target_buffer, handler &command_group_handler, const range< Dimensions > &offset, const range< Dimensions > &range)

  Construct a buffer accessor from a buffer given a specific range for access permissions and an offset that provides the starting point for the access range using a command group handler object from the command group scope.
- accessor (const range< Dimensions > &allocation_size, handler &command_group_handler)

  Construct an accessor of dimension Dimensions with elements of type DataType using the passed range to specify the size in each dimension.
- auto get_range () const

  Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.
- auto get_count () const

  Returns the total number of elements behind the accessor.
- auto get_size () const

  Returns the size of the underlying buffer storage in bytes.
- accessor_detail::reference operator[ ] (std::size_t index)

  Use the accessor with integers à la [][][].
- accessor_detail::reference operator[ ] (std::size_t index) const

  Use the accessor with integers à la [][][].
- auto & operator[ ] (id< dimensionality > index)

  To use the accessor with [id<>].
- auto & operator[ ] (id< dimensionality > index) const

  To use the accessor with [id<>].
- auto & operator[ ] (item< dimensionality > index)

  To use an accessor with [item<>].
- auto & operator[ ] (item< dimensionality > index) const

  To use an accessor with [item<>].
- auto & operator[ ] (nd_item< dimensionality > index)

  To use an accessor with an [nd_item<>].

- auto & operator[] (nd_item< dimensionality > index) const

  To use an accessor with an [nd_item<>].
- accessor_detail::reference operator* ()

  Get the first element of the accessor.
- accessor_detail::reference operator* () const

  Get the first element of the accessor.
- auto get_pointer () const

  Get the pointer to the start of the data.
- accessor_detail::iterator begin () const

  Forward all the iterator functions to the implementation.
- accessor_detail::iterator end () const
- accessor_detail::const_iterator cbegin () const
- accessor_detail::const_iterator cend () const
- accessor_detail::reverse_iterator rbegin () const
- accessor_detail::reverse_iterator rend () const
- accessor_detail::const_reverse_iterator crbegin () const
- accessor_detail::const_reverse_iterator crend () const

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Types

- using accessor_detail = typename detail::accessor< DataType, Dimensions, AccessMode, Target >
- using implementation_t = typename accessor::shared_ptr_implementation

Private Attributes

- friend implementation_t

Additional Inherited Members

8.1.2.3.1 Member Typedef Documentation

8.1.2.3.1.1 accessor_detail

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor_detail = typename
detail::accessor<DataType, Dimensions, AccessMode, Target>  [private]
```

Definition at line 68 of file accessor.hpp.

8.1.2.3.1.2 implementation_t

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
using cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::implementation_t =
typename accessor::shared_ptr_implementation  [private]
```

Definition at line 71 of file accessor.hpp.

8.1.2.3.2 Constructor & Destructor Documentation

8.1.2.3.2.1 accessor() [1/4]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
template<typename Allocator >
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (
            buffer< DataType, Dimensions, Allocator > & target_buffer,
            handler & command_group_handler )  [inline]
```

Construct a buffer accessor from a buffer using a command group handler object from the command group scope.

Constructor only available for global_buffer or constant_buffer target.

access_target defines the form of access being obtained.

Todo Add template allocator type in all the accessor constructors in the specification or just use a more opaque Buffer type?

Todo fix specification where access mode should be target instead

Definition at line 97 of file accessor.hpp.

References cl::sycl::access::constant_buffer, cl::sycl::access::global_buffer, cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation, and cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00098                                          : implementation_t {
00099     new detail::accessor<DataType, Dimensions, AccessMode, Target> {
00100       target_buffer.implementation->implementation, command_group_handler }
00101   } {
00102     static_assert(Target == access::target::global_buffer
00103                   || Target == access::target::constant_buffer,
00104                   "access target should be global_buffer or constant_buffer "
00105                   "when a handler is used");
00106     // Now the implementation is created, register it
00107     implementation->register_accessor();
00108   }
```

8.1.2.3.2.2 accessor() [2/4]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
template<typename Allocator >
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (
            buffer< DataType, Dimensions, Allocator > & target_buffer )  [inline]
```

Construct a buffer accessor from a buffer.

Constructor only available for host_buffer target.

access_target defines the form of access being obtained.

Definition at line 118 of file accessor.hpp.

References cl::sycl::access::host_buffer, and cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00119     : implementation_t {
00120     new detail::accessor<DataType, Dimensions, AccessMode, Target> {
00121       target_buffer.implementation->implementation }
00122   } {
00123     static_assert(Target == access::target::host_buffer,
00124                   "without a handler, access target should be host_buffer");
00125   }
```

8.1.2.3.2.3 accessor() [3/4]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
template<typename Allocator >
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (
            buffer< DataType, Dimensions, Allocator > & target_buffer,
            handler & command_group_handler,
            const range< Dimensions > & offset,
            const range< Dimensions > & range )  [inline]
```

Construct a buffer accessor from a buffer given a specific range for access permissions and an offset that provides the starting point for the access range using a command group handler object from the command group scope.

This accessor limits the processing of the buffer to the [offset, offset+range[for every dimension. Any other parts of the buffer will be unaffected.

Constructor only available for access modes global_buffer, and constant_buffer (see Table "Buffer accessor constructors"). access_target defines the form of access being obtained.

This accessor is recommended for discard-write and discard read write access modes, when the unaffected parts of the processing should be retained.

Definition at line 146 of file accessor.hpp.

References cl::sycl::detail::unimplemented().

```
00149                                         {
00150     detail::unimplemented();
00151   }
```

Here is the call graph for this function:

8.1.2.3.2.4 accessor() [4/4]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor (
            const range< Dimensions > & allocation_size,
            handler & command_group_handler )  [inline]
```

Construct an accessor of dimension Dimensions with elements of type DataType using the passed range to specify the size in each dimension.

It needs as a parameter a command group handler object from the command group scope. Constructor only available if AccessMode is local, see Table 3.25.

Definition at line 162 of file accessor.hpp.

References cl::sycl::access::local.

```
00164     : implementation_t { new detail::accessor<DataType,
00165                                                Dimensions,
00166                                                AccessMode,
00167                                                access::target::local> {
00168       allocation_size, command_group_handler
00169         }
00170   }
00171   {
00172     static_assert(Target == access::target::local,
00173                   "This accessor constructor requires "
00174                   "access target be local");
00175   }
```

8.1.2.3.3 Member Function Documentation

8.1.2.3.3.1 begin()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >↩
::begin ( ) const  [inline]
```

Forward all the iterator functions to the implementation.

Todo Add these functions to the specification

Todo The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

Todo try to solve it by using some enable_if on array constness?

Todo The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Todo Factor out these in a template helper

Definition at line 346 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00346                                                   {
00347     return implementation->begin();
00348   }
```

8.1.2.3.3.2 cbegin()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::const_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target
>::cbegin ( ) const  [inline]
```

Definition at line 363 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00363                                                       {
00364     return implementation->cbegin();
00365   }
```

8.1.2.3.3.3 cend()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::const_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target
>::cend ( ) const  [inline]
```

Definition at line 368 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00368                                                    {
00369     return implementation->cend();
00370   }
```

8.1.2.3.3.4 crbegin()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::const_reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode,
Target >::crbegin ( ) const  [inline]
```

Definition at line 389 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00389                                                          {
00390     return implementation->rbegin();
00391   }
```

8.1.2.3.3.5 crend()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::const_reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode,
Target >::crend ( ) const  [inline]
```

Definition at line 394 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00394                                                          {
00395     return implementation->rend();
00396   }
```

8.1.2.3.3.6 end()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::end
( ) const [inline]
```

Definition at line 352 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00352                                                {
00353     return implementation->end();
00354   }
```

8.1.2.3.3.7 get_count()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_count ( ) const
[inline]
```

Returns the total number of elements behind the accessor.

Equal to get_range()[0] ∗ ... ∗ get_range()[Dimensions-1].

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩ ://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 206 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00206                                 {
00207     return implementation->get_count();
00208   }
```

8.1.2.3.3.8 get_pointer()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_pointer ( ) const
[inline]
```

Get the pointer to the start of the data.

Todo Should it be named data() instead?

Definition at line 318 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00318                                 {
00319     return implementation->get_pointer();
00320   }
```

8.1.2.3.3.9 get_range()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_range ( ) const
[inline]
```

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 186 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00186                         {
00187     /* Interpret the shape which is a pointer to the first element as an
00188        array of Dimensions elements so that the range<Dimensions>
00189        constructor is happy with this collection
00190
00191        \todo Add also a constructor in range<> to accept a const
00192        std::size_t *?
00193     */
00194     return implementation->get_range();
00195   }
```

8.1.2.3.3.10 get_size()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::get_size ( ) const  [inline]
```

Returns the size of the underlying buffer storage in bytes.

Todo It is incompatible with buffer get_size() in the spec

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 219 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00219                         {
00220     return implementation->get_size();
00221   }
```

8.1.2.3.3.11 operator*() [1/2]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >↩
::operator* ( )  [inline]
```

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification

Definition at line 294 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00294                                                {
00295     return **implementation;
00296   }
```

8.1.2.3.3.12 operator*() [2/2]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >↩
::operator* ( ) const  [inline]
```

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification?

Todo Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Definition at line 309 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00309                                                        {
00310     return **implementation;
00311   }
```

8.1.2.3.3.13 operator[]() [1/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >↩
::operator[] (
            std::size_t index )  [inline]
```

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 229 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00229                                                                    {
00230 //#ifdef TRISYCL_DEVICE
00231 //#else
00232     return (*implementation)[index];
00233 //#endif
00234   }
```

8.1.2.3.3.14 operator[]() [2/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::reference cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >↩
::operator[] (
            std::size_t index ) const  [inline]
```

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 242 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00242                                                                    {
00243     return (*implementation)[index];
00244   }
```

8.1.2.3.3.15 operator[]() [3/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (
            id< dimensionality > index )  [inline]
```

To use the accessor with [id<>].

Definition at line 248 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00248                                                {
00249     return (*implementation)[index];
00250   }
```

8.1.2.3.3.16 operator[]() [4/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (
            id< dimensionality > index ) const  [inline]
```

To use the accessor with [id<>].

Definition at line 254 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00254                                                      {
00255     return (*implementation)[index];
00256   }
```

8.1.2.3.3.17 operator[]() [5/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (
            item< dimensionality > index )  [inline]
```

To use an accessor with [item<>].

Definition at line 260 of file accessor.hpp.

References cl::sycl::item< Dimensions >::get_id().

```
00260                                                 {
00261     return (*this)[index.get_id()];
00262   }
```

Here is the call graph for this function:

8.1.2.3.3.18 operator[]() [6/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (
            item< dimensionality > index ) const  [inline]
```

To use an accessor with [item<>].

Definition at line 266 of file accessor.hpp.

References cl::sycl::item< Dimensions >::get_id().

```
00266                                                    {
00267     return (*this)[index.get_id()];
00268   }
```

Here is the call graph for this function:

8.1.2.3.3.19 operator[]() [7/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (
            nd_item< dimensionality > index )  [inline]
```

To use an accessor with an [nd_item<>].

Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 275 of file accessor.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

```
00275                                                   {
00276     return (*this)[index.get_global()];
00277   }
```

Here is the call graph for this function:

8.1.2.3.3.20 operator[]() [8/8]

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
auto& cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[] (
            nd_item< dimensionality > index ) const  [inline]
```

To use an accessor with an [nd_item<>].

Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 283 of file accessor.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

```
00283                                                   {
00284     return (*this)[index.get_global()];
00285   }
```

Here is the call graph for this function:

8.1.2.3.3.21 rbegin()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target
>::rbegin ( ) const  [inline]
```

Definition at line 373 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00373                                                    {
00374     return implementation->rbegin();
00375   };
```

8.1.2.3.3.22 rend()

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
accessor_detail::reverse_iterator cl::sycl::accessor< DataType, Dimensions, AccessMode, Target
>::rend ( ) const  [inline]
```

Definition at line 378 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< accessor< DataType, Dimensions, AccessMode, Target >, detail::accessor< DataType, Dimensions, AccessMode, Target > >::implementation.

```
00378                                                    {
00379     return implementation->rend();
00380   }
```

8.1.2.3.4 Member Data Documentation

8.1.2.3.4.1 dimensionality

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
constexpr auto cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::dimensionality
= Dimensions  [static]
```

Todo in the specification: store the dimension for user request

Definition at line 61 of file accessor.hpp.

8.1.2.3.4.2 implementation_t

```
template<typename DataType , int Dimensions, access::mode AccessMode, access::target Target =
access::target::global_buffer>
friend cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::implementation_t  [private]
```

Definition at line 74 of file accessor.hpp.

8.1.2.4 class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

The pipe accessor abstracts the way pipe data are accessed inside a kernel.

A specialization for an non-blocking pipe

Definition at line 408 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >:

Collaboration diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >:

Public Types

- using accessor_detail = detail::pipe_accessor< DataType, AccessMode, access::target::pipe >

Public Member Functions

- accessor (pipe< DataType > &p, handler &command_group_handler)

  Construct a pipe accessor from a pipe using a command group handler object from the command group scope.
- pipe_reservation< accessor > reserve (std::size_t size) const

  Make a reservation inside the pipe.
- auto & get_pipe_detail ()

  Get the underlying pipe implementation.

Additional Inherited Members

8.1.2.4.1 Member Typedef Documentation

8.1.2.4.1.1 accessor_detail

```
template<typename DataType , access::mode AccessMode>
using cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::accessor_detail =
detail::pipe_accessor<DataType, AccessMode, access::target::pipe>
```

Definition at line 413 of file accessor.hpp.

8.1.2.4.2 Constructor & Destructor Documentation

8.1.2.4.2.1 accessor()

```
template<typename DataType , access::mode AccessMode>
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::accessor (
            pipe< DataType > & p,
            handler & command_group_handler )  [inline]
```

Construct a pipe accessor from a pipe using a command group handler object from the command group scope.

access_target defines the form of access being obtained.

Definition at line 422 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation.

```
00423     : accessor_detail { p.implementation, command_group_handler } { }
```

8.1.2.4.3 Member Function Documentation

8.1.2.4.3.1 get_pipe_detail()

```
template<typename DataType , access::mode AccessMode>
auto& cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::get_pipe_detail ( )
[inline]
```

Get the underlying pipe implementation.

Definition at line 432 of file accessor.hpp.

References cl::sycl::get_pipe_detail().

```
00432                           {
00433     return accessor_detail::get_pipe_detail();
00434   }
```

Here is the call graph for this function:

8.1.2.4.3.2 reserve()

```
template<typename DataType , access::mode AccessMode>
pipe_reservation<accessor> cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe
>::reserve (
            std::size_t size ) const  [inline]
```

Make a reservation inside the pipe.

Definition at line 426 of file accessor.hpp.

```
00426                                                                    {
00427     return accessor_detail::reserve(size);
00428   }
```

8.1.2.5 class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

template<typename DataType, access::mode AccessMode>
class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

The pipe accessor abstracts the way pipe data are accessed inside a kernel.

A specialization for a blocking pipe

Definition at line 446 of file accessor.hpp.

Inheritance diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >:

Collaboration diagram for cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >:

Public Types

- using accessor_detail = detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >

Public Member Functions

- accessor (pipe< DataType > &p, handler &command_group_handler)

  Construct a pipe accessor from a pipe using a command group handler object from the command group scope.
- pipe_reservation< accessor > reserve (std::size_t size) const

  Make a reservation inside the pipe.
- auto & get_pipe_detail ()

  Get the underlying pipe implementation.

Additional Inherited Members

8.1.2.5.1 Member Typedef Documentation

8.1.2.5.1.1 accessor_detail

```
template<typename DataType , access::mode AccessMode>
using cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::accessor↩
_detail = detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
```

Definition at line 451 of file accessor.hpp.

8.1.2.5.2 Constructor & Destructor Documentation

8.1.2.5.2.1 accessor()

```
template<typename DataType , access::mode AccessMode>
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::accessor (
            pipe< DataType > & p,
            handler & command_group_handler )  [inline]
```

Construct a pipe accessor from a pipe using a command group handler object from the command group scope.

access_target defines the form of access being obtained.

Definition at line 460 of file accessor.hpp.

References cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation.

```
00461     : accessor_detail { p.implementation, command_group_handler } { }
```

8.1.2.5.3 Member Function Documentation

8.1.2.5.3.1 get_pipe_detail()

```
template<typename DataType , access::mode AccessMode>
auto& cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe↩
_detail ( )  [inline]
```

Get the underlying pipe implementation.

Definition at line 471 of file accessor.hpp.

References cl::sycl::get_pipe_detail().

```
00471                                {
00472     return accessor_detail::get_pipe_detail();
00473   }
```

Here is the call graph for this function:

8.1.2.5.3.2 reserve()

```
template<typename DataType , access::mode AccessMode>
pipe_reservation<accessor> cl::sycl::accessor< DataType, 1, AccessMode, access::target↩
::blocking_pipe >::reserve (
            std::size_t size ) const  [inline]
```

Make a reservation inside the pipe.

Definition at line 465 of file accessor.hpp.

```
00465                                                    {
00466     return accessor_detail::reserve(size);
00467   }
```

8.1.2.6 class cl::sycl::detail::accessor

template<typename T, int Dimensions, access::mode Mode, access::target Target>
class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way.

This implementation relies on boost::multi_array to provide this nice syntax and behaviour.

Right now the aim of this class is just to access to the buffer in a read-write mode, even if capturing the multi_array_ref from a lambda make it const (since in examples we have lambda with [=] without mutable lambda).

Todo Use the access::mode

Definition at line 39 of file local_accessor.hpp.

Inheritance diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, Target >:

Collaboration diagram for cl::sycl::detail::accessor< T, Dimensions, Mode, Target >:

Public Types

- using value_type = T
- using element = T
- using reference = typename array_view_type::reference
- using const_reference = typename array_view_type::const_reference
- using iterator = typename array_view_type::iterator

  Inherit the iterator types from the implementation.
- using const_iterator = typename array_view_type::const_iterator
- using reverse_iterator = typename array_view_type::reverse_iterator
- using const_reverse_iterator = typename array_view_type::const_reverse_iterator

Public Member Functions

- accessor (std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer)

  Construct a host accessor from an existing buffer.
- accessor (std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer, handler &command_group_↩ handler)

  Construct a device accessor from an existing buffer.
- void register_accessor ()

  Register the accessor once a `std::shared_ptr` is created on it.
- auto get_range () const

  Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.
- auto get_count () const

  Returns the total number of elements behind the accessor.
- auto get_size () const

  Returns the size of the underlying buffer storage in bytes.
- reference operator[ ] (std::size_t index)

  Use the accessor with integers à la [][][].
- reference operator[ ] (std::size_t index) const

  Use the accessor with integers à la [][][].
- auto & operator[ ] (id< dimensionality > index)

  To use the accessor with [id<>].
- auto & operator[ ] (id< dimensionality > index) const

  To use the accessor with [id<>].
- auto & operator[ ] (item< dimensionality > index)

  To use an accessor with [item<>].
- auto & operator[ ] (item< dimensionality > index) const

  To use an accessor with [item<>].
- auto & operator[ ] (nd_item< dimensionality > index)

  To use an accessor with an [nd_item<>].
- auto & operator[ ] (nd_item< dimensionality > index) const

  To use an accessor with an [nd_item<>].
- reference operator∗ ()

  Get the first element of the accessor.
- reference operator∗ () const

  Get the first element of the accessor.
- detail::buffer< T, Dimensions > & get_buffer ()

  Get the buffer used to create the accessor.
- constexpr bool is_read_access () const

Test if the accessor has a read access right.

- constexpr bool is_write_access () const

  Test if the accessor has a write access right.
- auto get_pointer () const

  Return the pointer to the data.
- iterator begin () const

  Forward all the iterator functions to the implementation.
- iterator end () const
- const_iterator cbegin () const
- const_iterator cend () const
- reverse_iterator rbegin () const
- reverse_iterator rend () const
- const_reverse_iterator crbegin () const
- const_reverse_iterator crend () const

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Types

- using array_view_type = boost::multi_array_ref< T, Dimensions >

  The implementation is a multi_array_ref wrapper.
- using writable_array_view_type = typename std::remove_const< array_view_type >::type

Private Member Functions

- boost::compute::buffer get_cl_buffer () const override

  Get the boost::compute::buffer or throw if unset.
- void copy_in_cl_buffer ()

  Lazily associate a CL buffer to the SYCL buffer and copy data in it if required, updates the state of the data in the buffer across contexts.
- void copy_back_cl_buffer ()

  Does nothing.

Private Attributes

- std::shared_ptr< detail::buffer< T, Dimensions > > buf

  Keep a reference to the accessed buffer.
- array_view_type array

  The way the buffer is really accessed.
- friend handler

Additional Inherited Members

8.1.2.6.1 Member Typedef Documentation

8.1.2.6.1.1 array_view_type

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array_view_type = boost↩
::multi_array_ref<T, Dimensions>  [private]
```

The implementation is a multi_array_ref wrapper.

Definition at line 78 of file accessor.hpp.

8.1.2.6.1.2 const_iterator

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::const_iterator = typename
array_view_type::const_iterator
```

Definition at line 113 of file accessor.hpp.

8.1.2.6.1.3 const_reference

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::const_reference = typename
array_view_type::const_reference
```

Definition at line 106 of file accessor.hpp.

8.1.2.6.1.4 const_reverse_iterator

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::const_reverse_iterator =
typename array_view_type::const_reverse_iterator
```

Definition at line 116 of file accessor.hpp.

8.1.2.6.1.5 element

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::element = T
```

Definition at line 104 of file accessor.hpp.

8.1.2.6.1.6 iterator

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::iterator = typename array_↩
view_type::iterator
```

Inherit the iterator types from the implementation.

Todo Add iterators to accessors in the specification

Definition at line 112 of file accessor.hpp.

8.1.2.6.1.7 reference

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::reference = typename array_↩
view_type::reference
```

Definition at line 105 of file accessor.hpp.

8.1.2.6.1.8 reverse_iterator

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::reverse_iterator = typename
array_view_type::reverse_iterator
```

Definition at line 114 of file accessor.hpp.

8.1.2.6.1.9 value_type

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::value_type = T
```

Todo in the specification: store the types for user request as STL or C++AMP

Definition at line 103 of file accessor.hpp.

8.1.2.6.1.10 writable_array_view_type

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
using cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::writable_array_view_type =
typename std::remove_const<array_view_type>::type  [private]
```

Definition at line 82 of file accessor.hpp.

8.1.2.6.2 Constructor & Destructor Documentation

8.1.2.6.2.1 accessor() [1/2]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (
            std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer )  [inline]
```

Construct a host accessor from an existing buffer.

Todo fix the specification to rename target that shadows template parm

Definition at line 124 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size(), cl::sycl::access::host_buffer, cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access(), and TRISYCL_DUMP_T.

```
00124                                                                    :
00125     buf { target_buffer }, array { target_buffer->access } {
00126     target_buffer->template track_access_mode<Mode>();
00127     TRISYCL_DUMP_T("Create a host accessor write = " <<
      is_write_access());
00128     static_assert(Target == access::target::host_buffer,
00129                   "without a handler, access target should be host_buffer");
00130     /* The host needs to wait for all the producers of the buffer to
00131        have finished */
00132     buf->wait();
00133
00134 #ifdef TRISYCL_OPENCL
00135     /* For the host context, we are obligated to update the buffer state
00136        during the accessors creation, otherwise we have no way of knowing
00137        if a buffer was modified on the host. This is only true because
00138        host accessors are blocking
00139      */
00140     cl::sycl::context ctx;
00141     buf->update_buffer_state(ctx, Mode, get_size(), array.data());
00142 #endif
00143   }
```

Here is the call graph for this function:

8.1.2.6.2.2 accessor() [2/2]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor (
            std::shared_ptr< detail::buffer< T, Dimensions >> target_buffer,
            handler & command_group_handler )  [inline]
```

Construct a device accessor from an existing buffer.

Todo fix the specification to rename target that shadows template parm

Definition at line 151 of file accessor.hpp.

References cl::sycl::detail::buffer_add_to_task(), cl::sycl::access::constant_buffer, cl::sycl::access::global_buffer, cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access(), and TRISYCL_DUMP_T.

```
                                                :
    buf { target_buffer }, array { target_buffer->access } {
    target_buffer->template track_access_mode<Mode>();
    TRISYCL_DUMP_T("Create a kernel accessor write = " <<
is_write_access());
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
    // Register the buffer to the task dependencies
    task = buffer_add_to_task(buf, &command_group_handler,
is_write_access());
  }
```

Here is the call graph for this function:

8.1.2.6.3 Member Function Documentation

8.1.2.6.3.1 begin()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::begin ( ) const  [inline]
```

Forward all the iterator functions to the implementation.

Todo Add these functions to the specification

Todo The fact that the lambda capture make a const copy of the accessor is not yet elegantly managed... The issue is that begin()/end() dispatch is made according to the accessor constness and not from the array member constness...

Todo try to solve it by using some enable_if on array constness?

Todo The issue is that the end may not be known if it is implemented by a raw OpenCL cl_mem... So only provide on the device the iterators related to the start? Actually the accessor needs to know a part of the shape to have the multidimentional addressing. So this only require a size_t more...

Todo Factor out these in a template helper

Todo Do we need this in detail::accessor too or only in accessor?

Definition at line 404 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::begin().

```
00404                                {
00405     return const_cast<writable_array_view_type &>(array).
      begin();
00406   }
```

Here is the caller graph for this function:

8.1.2.6.3.2 cbegin()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::cbegin ( ) const
[inline]
```

Definition at line 421 of file accessor.hpp.

```
00421 { return array.begin(); }
```

8.1.2.6.3.3 cend()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
const_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::cend ( ) const
[inline]
```

Definition at line 424 of file accessor.hpp.

```
00424 { return array.end(); }
```

8.1.2.6.3.4 copy_back_cl_buffer()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
void cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_back_cl_buffer ( )  [inline],
[private]
```

Does nothing.

Definition at line 477 of file accessor.hpp.

```
00477                                {
00478     /* The copy back is handled by the host accessor and the buffer destructor.
00479        We don't need to systematically transfer the data after the
00480        kernel execution
00481
00482        \todo Figure out what to do with this function
00483     */
00484   }
```

8.1.2.6.3.5 copy_in_cl_buffer()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
void cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer ( ) [inline],
[private]
```

Lazily associate a CL buffer to the SYCL buffer and copy data in it if required, updates the state of the data in the buffer across contexts.

Definition at line 467 of file accessor.hpp.

References cl::sycl::detail::task::get_queue(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get↩_size().

```
00467                                {
00468     /* Create the OpenCL buffer and copy in it the data from the host if
00469        the buffer doesn't already exists or if the data is not up to date
00470     */
00471     auto ctx = task->get_queue()->get_context();
00472     buf->update_buffer_state(ctx, Mode, get_size(), array.data());
00473   }
```

Here is the call graph for this function:

8.1.2.6.3.6 crbegin()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::crbegin ( )
const [inline]
```

Definition at line 445 of file accessor.hpp.

```
00445 { return array.rbegin(); }
```

8.1.2.6.3.7 crend()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
const_reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::crend ( )
const [inline]
```

Definition at line 448 of file accessor.hpp.

```
00448 { return array.rend(); }
```

8.1.2.6.3.8 end()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::end ( ) const  [inline]
```

Definition at line 410 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::end().

```
00410                                   {
00411     return const_cast<writable_array_view_type &>(array).
      end();
00412   }
```

Here is the caller graph for this function:

8.1.2.6.3.9 get_buffer()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
detail::buffer<T, Dimensions>& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >↩
::get_buffer ( )  [inline]
```

Get the buffer used to create the accessor.

Definition at line 333 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::buf.

```
00333                                                   {
00334     return *buf;
00335   }
```

8.1.2.6.3.10 get_cl_buffer()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
boost::compute::buffer cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_cl_↩
buffer ( ) const  [inline], [override], [private], [virtual]
```

Get the boost::compute::buffer or throw if unset.

Implements cl::sycl::detail::accessor_base.

Definition at line 457 of file accessor.hpp.

References cl::sycl::detail::task::get_queue().

```
00457                                                    {
00458     // This throws if not set
00459     auto ctx = task->get_queue()->get_context();
00460     return buf->get_cl_buffer(ctx);
00461   }
```

Here is the call graph for this function:

8.1.2.6.3.11 get_count()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count ( ) const  [inline]
```

Returns the total number of elements behind the accessor.

Equal to get_range()[0] ∗ ... ∗ get_range()[Dimensions-1].

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 229 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::get_size(), and cl::sycl↩::detail::accessor< T, Dimensions, Mode, Target >::get_size().

```
00229                                 {
00230     return array.num_elements();
00231   }
```

Here is the caller graph for this function:

8.1.2.6.3.12 get_pointer()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_pointer ( ) const  [inline]
```

Return the pointer to the data.

Todo Implement the various pointer address spaces

Definition at line 374 of file accessor.hpp.

```
00374                        {
00375     return array.data();
00376   }
```

8.1.2.6.3.13 get_range()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_range ( ) const  [inline]
```

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https↩ ://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 207 of file accessor.hpp.

```
00207                        {
00208     /* Interpret the shape which is a pointer to the first element as an
00209        array of Dimensions elements so that the range<Dimensions>
00210        constructor is happy with this collection
00211
00212        \todo Add also a constructor in range<> to accept a const
00213        std::size_t *?
00214     */
00215     return range<Dimensions> {
00216       *(const std::size_t (*)[Dimensions])(array.shape())
00217         };
00218   }
```

8.1.2.6.3.14 get_size()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_size ( ) const  [inline]
```

Returns the size of the underlying buffer storage in bytes.

Todo Move on https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404

Definition at line 240 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_count().

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer().

```
00240                          {
00241     return get_count()*sizeof(value_type);
00242   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.6.3.15 is_read_access()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_read_access ( )
const [inline]
```

Test if the accessor has a read access right.

Todo Strangely, it is not really constexpr because it is not a static method...

Todo to move in the access::mode enum class and add to the specification ?

Definition at line 346 of file accessor.hpp.

References cl::sycl::access::discard_read_write, cl::sycl::access::read, and cl::sycl::access::read_write.

```
00346                                        {
00347     return Mode == access::mode::read
00348       || Mode == access::mode::read_write
00349       || Mode == access::mode::discard_read_write;
00350   }
```

8.1.2.6.3.16 is_write_access()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
constexpr bool cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::is_write_access ( )
const [inline]
```

Test if the accessor has a write access right.

Todo Strangely, it is not really constexpr because it is not a static method...

Todo to move in the access::mode enum class and add to the specification ?

Definition at line 361 of file accessor.hpp.

References cl::sycl::access::discard_read_write, cl::sycl::access::discard_write, cl::sycl::access::read_write, and cl::sycl::access::write.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor().

```
00361                                        {
00362     return Mode == access::mode::write
00363       || Mode == access::mode::read_write
00364       || Mode == access::mode::discard_write
00365       || Mode == access::mode::discard_read_write;
00366   }
```

Here is the caller graph for this function:

8.1.2.6.3.17 operator*() [1/2]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* ( )  [inline]
```

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification

Definition at line 312 of file accessor.hpp.

```
00312                        {
00313     return *array.data();
00314   }
```

8.1.2.6.3.18 operator*() [2/2]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator* ( ) const  [inline]
```

Get the first element of the accessor.

Useful with an accessor on a scalar for example.

Todo Add in the specification?

Todo Add the concept of 0-dim buffer and accessor for scalar and use an implicit conversion to value_type reference to access the value with the accessor?

Definition at line 327 of file accessor.hpp.

```
00327                              {
00328     return *array.data();
00329   }
```

8.1.2.6.3.19 operator[]() [1/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            std::size_t index )  [inline]
```

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 250 of file accessor.hpp.

```
00250                                      {
00251     return array[index];
00252   }
```

8.1.2.6.3.20 operator[]() [2/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
reference cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            std::size_t index ) const  [inline]
```

Use the accessor with integers à la [][][].

Use array_view_type::reference instead of auto& because it does not work in some dimensions.

Definition at line 260 of file accessor.hpp.

```
00260                                                {
00261     return array[index];
00262   }
```

8.1.2.6.3.21 operator[]() [3/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            id< dimensionality > index )  [inline]
```

To use the accessor with [id<>].

Definition at line 266 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
00266                                                {
00267     return array(index);
00268   }
```

8.1.2.6.3.22 operator[]() [4/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            id< dimensionality > index ) const  [inline]
```

To use the accessor with [id<>].

Definition at line 272 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

```
00272                                                {
00273     return array(index);
00274   }
```

8.1.2.6.3.23 operator[]() [5/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            item< dimensionality > index )  [inline]
```

To use an accessor with [item<>].

Definition at line 278 of file accessor.hpp.

```
00278                                                {
00279     return (*this)[index.get()];
00280   }
```

8.1.2.6.3.24 operator[]() [6/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            item< dimensionality > index ) const  [inline]
```

To use an accessor with [item<>].

Definition at line 284 of file accessor.hpp.

```
00284                                                      {
00285     return (*this)[index.get()];
00286   }
```

8.1.2.6.3.25 operator[]() [7/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            nd_item< dimensionality > index )  [inline]
```

To use an accessor with an [nd_item<>].

Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 293 of file accessor.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

```
00293                                                    {
00294     return (*this)[index.get_global()];
00295   }
```

Here is the call graph for this function:

8.1.2.6.3.26 operator[]() [8/8]

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
auto& cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[] (
            nd_item< dimensionality > index ) const  [inline]
```

To use an accessor with an [nd_item<>].

Todo Add in the specification because used by HPC-GPU slide 22

Definition at line 301 of file accessor.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

```
00301                                                    {
00302     return (*this)[index.get_global()];
00303   }
```

Here is the call graph for this function:

8.1.2.6.3.27 rbegin()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rbegin ( ) const
[inline]
```

Definition at line 428 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::rbegin().

```
00428                                    {
00429     return const_cast<writable_array_view_type &>(array).
      rbegin();
00430   }
```

Here is the caller graph for this function:

8.1.2.6.3.28 register_accessor()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
void cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor ( )  [inline]
```

Register the accessor once a `std::shared_ptr` is created on it.

This is to be called from outside once the object is created. It has been tried directly inside the contructor, but calling `shared_from_this()` from the constructor dead-lock with libstdc++6

Todo Double-check with the C++ committee on this issue.

Definition at line 174 of file accessor.hpp.

References cl::sycl::detail::task::add_postlude(), cl::sycl::detail::task::add_prelude(), cl::sycl::detail::task::get_queue(), cl::sycl::detail::task::register_accessor(), and cl::sycl::detail::accessor_base::set_order().

```
00174                                {
00175     if (!task->get_queue()->is_host()) {
00176       // To keep alive this accessor in the following lambdas
00177       auto acc = this->shared_from_this();
00178       // Attach the accessor to the task and get its order
00179       set_order(task->register_accessor(acc));
00180 #ifdef TRISYCL_OPENCL
00181       /* Before running the kernel, make sure the cl_mem behind this
00182          accessor is up-to-date on the device if needed and pass it to
00183          the kernel */
00184       task->add_prelude([=] {
00185           acc->copy_in_cl_buffer();
00186         });
00187       // After running the kernel, deal with some copy-back if needed
00188       task->add_postlude([=] {
00189           /* Even if this function does nothing, it is required to
00190              have the capture of acc to keep the accessor alive across
00191              the kernel execution up to the execution postlude */
00192           acc->copy_back_cl_buffer();
00193         });
00194 #endif
00195     }
00196   }
```

Here is the call graph for this function:

8.1.2.6.3.29 rend()

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
reverse_iterator cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rend ( ) const
[inline]
```

Definition at line 434 of file accessor.hpp.

References cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::rend().

```
00434                                   {
00435     return const_cast<writable_array_view_type &>(array).
      rend();
00436   }
```

Here is the caller graph for this function:

8.1.2.6.4 Member Data Documentation

8.1.2.6.4.1 array

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
array_view_type cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::array  [mutable],
[private]
```

The way the buffer is really accessed.

Use a mutable member because the accessor needs to be captured by value in the lambda which is then read-only. This is to avoid the user to use mutable lambda or have a lot of const_cast as previously done in this implementation

Definition at line 91 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::begin(), cl::sycl::detail↩ ::accessor< T, Dimensions, Mode, Target >::begin(), cl::sycl::detail::accessor< T, Dimensions, Mode, access↩ ::target::local >::end(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::end(), cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[ ](), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[ ](), cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::rbegin(), cl::sycl::detail↩ ::accessor< T, Dimensions, Mode, Target >::rbegin(), cl::sycl::detail::accessor< T, Dimensions, Mode, access↩ ::target::local >::rend(), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::rend().

8.1.2.6.4.2 buf

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
std::shared_ptr<detail::buffer<T, Dimensions> > cl::sycl::detail::accessor< T, Dimensions,
Mode, Target >::buf [private]
```

Keep a reference to the accessed buffer.

Beware that it owns the buffer, which means that the accessor has to be destroyed to release the buffer and potentially unblock a kernel at the end of its execution

Definition at line 75 of file accessor.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::get_buffer().

8.1.2.6.4.3 dimensionality

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
constexpr auto cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::dimensionality =
Dimensions [static]
```

Todo in the specification: store the dimension for user request

Todo Use another name, such as from C++17 committee discussions.

Definition at line 99 of file accessor.hpp.

8.1.2.6.4.4 handler

```
template<typename T , int Dimensions, access::mode Mode, access::target Target>
friend cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::handler [private]
```

Definition at line 454 of file accessor.hpp.

8.1.2.7 class cl::sycl::detail::buffer

template<typename T, int Dimensions = 1>
class cl::sycl::detail::buffer< T, Dimensions >

A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on.

In the case we initialize it from a pointer, for now we just wrap the data with boost::multi_array_ref to provide the VLA semantics without any storage.

Definition at line 36 of file accessor.hpp.

Inheritance diagram for cl::sycl::detail::buffer< T, Dimensions >:

Collaboration diagram for cl::sycl::detail::buffer< T, Dimensions >:

Public Types

- using element = T
- using value_type = T
- using non_const_value_type = std::remove_const_t< value_type >

Public Member Functions

- buffer (const range< Dimensions > &r)

  Create a new read-write buffer of size.
- buffer (T *host_data, const range< Dimensions > &r)

  Create a new read-write buffer from.
- template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>::value>>
  buffer (const T *host_data, const range< Dimensions > &r)

  Create a new read-only buffer from.
- buffer (shared_ptr_class< T > &host_data, const range< Dimensions > &r)

  Create a new buffer with associated memory, using the data in host_data.
- template<typename Iterator >
  buffer (Iterator start_iterator, Iterator end_iterator)

  Create a new allocated 1D buffer from the given elements.
- ~buffer ()

  Create a new sub-buffer without allocation to have separate accessors later.
- void mark_as_written ()

  Enforce the buffer to be considered as being modified.
- template<access::mode Mode, access::target Target = access::target::host_buffer>
  void track_access_mode ()

  This method is to be called whenever an accessor is created.

- auto get_range () const

  Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.
- auto get_count () const

  Returns the total number of elements in the buffer.
- auto get_size () const

  Returns the size of the buffer storage in bytes.
- void set_final_data (std::weak_ptr< T > &&final_data)

  Set the weak pointer as destination for write-back on buffer destruction.
- void set_final_data (std::shared_ptr< T > &&final_data)

  Provide destination for write-back on buffer destruction as a shared pointer.
- void set_final_data (std::nullptr_t)

  Disable write-back on buffer destruction as an iterator.
- template<typename Iterator >
  void set_final_data (Iterator final_data)

  Provide destination for write-back on buffer destruction as an iterator.
- boost::optional< std::future< void > > get_destructor_future ()

  Get a `future` to wait from inside the `cl::sycl::buffer` in case there is something to copy back to the host.

Private Member Functions

- auto allocate_buffer (const range< Dimensions > &r)

  Allocate uninitialized buffer memory.
- void deallocate_buffer ()

  Deallocate buffer memory if required.
- template<typename BaseType = T, typename DataType >
  void call_update_buffer_state (cl::sycl::context ctx, access::mode mode, size_t size, DataType *data, std↩::enable_if_t<!std::is_const< BaseType > ::value > *=0)

  Function pair to work around the fact that T might be a `const` type.
- template<typename BaseType = T, typename DataType >
  void call_update_buffer_state (cl::sycl::context ctx, access::mode mode, size_t size, DataType *data, std↩::enable_if_t< std::is_const< BaseType > ::value > *=0)

  Version of `call_update_buffer_state` that does nothing.

Private Attributes

- std::allocator< non_const_value_type > alloc

  The allocator to be used when some memory is needed.
- boost::multi_array_ref< value_type, Dimensions > access

  This is the multi-dimensional interface to the data that may point to either allocation in the case of storage managed by SYCL itself or to some other memory location in the case of host memory or storage<> abstraction use.
- non_const_value_type * allocation = nullptr

  If some allocation is requested on the host for the buffer memory, this is where the memory is attached to.
- boost::optional< std::function< void(void)> > final_write_back
- shared_ptr_class< T > input_shared_pointer
- bool data_host = false
- bool copy_if_modified = false
- bool modified = false

Friends

- template<typename U , int D, access::mode Mode, access::target Target>
  class detail::accessor

Additional Inherited Members

8.1.2.7.1 Member Typedef Documentation

8.1.2.7.1.1 element

```
template<typename T, int Dimensions = 1>
using cl::sycl::detail::buffer< T, Dimensions >::element = T
```

Definition at line 49 of file buffer.hpp.

8.1.2.7.1.2 non_const_value_type

```
template<typename T, int Dimensions = 1>
using cl::sycl::detail::buffer< T, Dimensions >::non_const_value_type = std::remove_const_↩
t<value_type>
```

Definition at line 53 of file buffer.hpp.

8.1.2.7.1.3 value_type

```
template<typename T, int Dimensions = 1>
using cl::sycl::detail::buffer< T, Dimensions >::value_type = T
```

Definition at line 50 of file buffer.hpp.

8.1.2.7.2 Constructor & Destructor Documentation

8.1.2.7.2.1 buffer() [1/5]

```
template<typename T, int Dimensions = 1>
cl::sycl::detail::buffer< T, Dimensions >::buffer (
            const range< Dimensions > & r )  [inline]
```

Create a new read-write buffer of size.

Parameters

r	

Definition at line 107 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer().

```
00107 : access { allocate_buffer(r) } {}
```

Here is the call graph for this function:

8.1.2.7.2.2 buffer() [2/5]

```
template<typename T, int Dimensions = 1>
cl::sycl::detail::buffer< T, Dimensions >::buffer (
            T * host_data,
            const range< Dimensions > & r )  [inline]
```

Create a new read-write buffer from.

Parameters

host_data	of size
r	without further allocation

Definition at line 112 of file buffer.hpp.

```
00112                                                   :
00113     access { host_data, r },
00114     data_host { true }
00115   {}
```

8.1.2.7.2.3 buffer() [3/5]

```
template<typename T, int Dimensions = 1>
template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>↩
::value>>
cl::sycl::detail::buffer< T, Dimensions >::buffer (
            const T * host_data,
            const range< Dimensions > & r )  [inline]
```

Create a new read-only buffer from.

Parameters

host_data	of size
r	without further allocation

If the buffer is non const, use a copy-on-write mechanism with internal writable memory.

Todo Clarify the semantics in the spec. What happens if the host change the host_data after buffer creation?

Only enable this constructor if the value type is not constant, because if it is constant, the buffer is constant too.

Definition at line 132 of file buffer.hpp.

```
00132                                                          :
00133     /* The buffer is read-only, even if the internal multidimensional
00134        wrapper is not. If a write accessor is requested, there should
00135        be a copy on write. So this pointer should not be written and
00136        this const_cast should be acceptable. */
00137     access { const_cast<T *>(host_data), r },
00138     data_host { true },
00139     /* Set copy_if_modified to true, so that if an accessor with write
00140        access is created, data are copied before to be modified. */
00141     copy_if_modified { true }
00142   {}
```

8.1.2.7.2.4 buffer() [4/5]

```
template<typename T, int Dimensions = 1>
cl::sycl::detail::buffer< T, Dimensions >::buffer (
            shared_ptr_class< T > & host_data,
            const range< Dimensions > & r )  [inline]
```

Create a new buffer with associated memory, using the data in host_data.

The ownership of the host_data is shared between the runtime and the user. In order to enable both the user application and the SYCL runtime to use the same pointer, a cl::sycl::mutex_class is used.

Definition at line 153 of file buffer.hpp.

```
00153                                                                  :
00154     access { host_data.get(), r },
00155     input_shared_pointer { host_data },
00156     data_host { true }
00157   {}
```

8.1.2.7.2.5 buffer() [5/5]

```
template<typename T, int Dimensions = 1>
template<typename Iterator >
cl::sycl::detail::buffer< T, Dimensions >::buffer (
            Iterator start_iterator,
            Iterator end_iterator )  [inline]
```

Create a new allocated 1D buffer from the given elements.

Definition at line 162 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer().

```
00162                                                    :
00163     access { allocate_buffer(std::distance(start_iterator, end_iterator)) }
00164     {
00165       /* Then assign allocation since this is the only multi_array
00166          method with this iterator interface */
00167       access.assign(start_iterator, end_iterator);
00168     }
```

Here is the call graph for this function:

8.1.2.7.2.6 ~buffer()

```
template<typename T, int Dimensions = 1>
cl::sycl::detail::buffer< T, Dimensions >::~buffer ( )  [inline]
```

Create a new sub-buffer without allocation to have separate accessors later.

Todo To implement and deal with reference counting buffer(buffer<T, Dimensions> b, index<Dimensions> base↩ _index, range<Dimensions> sub_range)

Todo Allow CLHPP objects too?

The buffer content may be copied back on destruction to some final location

Definition at line 191 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state(), cl::sycl::detail::buffer< T, Dimensions >::deallocate_buffer(), and cl::sycl::access::read.

```
            {
#ifdef TRISYCL_OPENCL
    /* We ensure that the host has the most up-to-date version of the data
       before the buffer is destroyed. This is necessary because we do not
       systematically transfer the data back from a device with
       \c copy_back_cl_buffer any more.
       \todo Optimize for the case the buffer is not based on host memory
    */
    cl::sycl::context ctx;
    auto size = access.num_elements() * sizeof(value_type);
    call_update_buffer_state(ctx, access::mode::read, size,
  access.data());

#endif
    if (modified && final_write_back)
      (*final_write_back)();
    // Allocate explicitly allocated memory if required
    deallocate_buffer();
  }
```

Here is the call graph for this function:

8.1.2.7.3 Member Function Documentation

8.1.2.7.3.1 allocate_buffer()

```
template<typename T, int Dimensions = 1>
auto cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer (
            const range< Dimensions > & r )  [inline], [private]
```

Allocate uninitialized buffer memory.

Definition at line 343 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::allocation, and cl::sycl::range< Dimensions >::get_count().

Referenced by cl::sycl::detail::buffer< T, Dimensions >::buffer(), and cl::sycl::detail::buffer< T, Dimensions >↩ ::track_access_mode().

```
                                                {
    auto count = r.get_count();
    // Allocate uninitialized memory
    allocation = alloc.allocate(count);
    return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
  }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.7.3.2 call_update_buffer_state() [1/2]

```
template<typename T, int Dimensions = 1>
template<typename BaseType = T, typename DataType >
void cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state (
            cl::sycl::context ctx,
            access::mode mode,
            size_t size,
            DataType * data,
            std::enable_if_t<!std::is_const< BaseType > ::value > *  = 0 )  [inline], [private]
```

Function pair to work around the fact that T might be a `const` type.

We call update_buffer_state only if T is not `const`, we have to use `enable_if` otherwise the compiler will try to cast `const void*` to `void*` if we create a buffer with a `const` type

Todo Use `if constexpr` when it is available with C++17

Definition at line 366 of file buffer.hpp.

References cl::sycl::detail::buffer_base::update_buffer_state().

Referenced by cl::sycl::detail::buffer< T, Dimensions >::~buffer().

```
00369                                                {
00370     update_buffer_state(ctx, mode, size, data);
00371   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.7.3.3 call_update_buffer_state() [2/2]

```
template<typename T, int Dimensions = 1>
template<typename BaseType = T, typename DataType >
void cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state (
            cl::sycl::context ctx,
            access::mode mode,
            size_t size,
            DataType * data,
            std::enable_if_t< std::is_const< BaseType > ::value > *  = 0 )  [inline], [private]
```

Version of `call_update_buffer_state` that does nothing.

It is called if the type of the data in the buffer is `const`

Definition at line 378 of file buffer.hpp.

```
00381                                                { }
```

8.1.2.7.3.4 deallocate_buffer()

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::deallocate_buffer ( )  [inline], [private]
```

Deallocate buffer memory if required.

Definition at line 352 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::~buffer().

```
00352                                {
00353     if (allocation)
00354       alloc.deallocate(allocation, access.num_elements());
00355   }
```

Here is the caller graph for this function:

8.1.2.7.3.5 get_count()

```
template<typename T, int Dimensions = 1>
auto cl::sycl::detail::buffer< T, Dimensions >::get_count ( ) const  [inline]
```

Returns the total number of elements in the buffer.

Equal to get_range()[0] ∗ ... ∗ get_range()[Dimensions-1].

Definition at line 280 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::get_size().

```
00280                               {
00281     return access.num_elements();
00282   }
```

Here is the caller graph for this function:

8.1.2.7.3.6 get_destructor_future()

```
template<typename T, int Dimensions = 1>
boost::optional<std::future<void> > cl::sycl::detail::buffer< T, Dimensions >::get_destructor↩
_future ( ) [inline]
```

Get a `future` to wait from inside the `cl::sycl::buffer` in case there is something to copy back to the host.

Returns

A `future` in the `optional` if there is something to wait for, otherwise an empty `optional`

Todo Make the function private again

Definition at line 393 of file buffer.hpp.

References cl::sycl::detail::buffer_base::notify_buffer_destructor.

```
00393                                                        {
00394     /* If there is only 1 shared_ptr user of the buffer, this is the
00395        caller of this function, the \c buffer_waiter, so there is no
00396        need to get a \ future otherwise there will be a dead-lock if
00397        there is only 1 thread waiting for itself.
00398
00399        Since \c use_count() is applied to a \c shared_ptr just created
00400        for this purpose, it actually increase locally the count by 1,
00401        so check for 1 + 1 use count instead...
00402     */
00403     // If the buffer's destruction triggers a write-back, wait
00404     if ((shared_from_this().use_count() > 2) &&
00405         modified && (final_write_back || data_host)) {
00406       // Create a promise to wait for
00407       notify_buffer_destructor = std::promise<void> {};
00408       // And return the future to wait for it
00409       return notify_buffer_destructor->get_future();
00410     }
00411     return boost::none;
00412   }
```

8.1.2.7.3.7 get_range()

```
template<typename T, int Dimensions = 1>
auto cl::sycl::detail::buffer< T, Dimensions >::get_range ( ) const [inline]
```

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Definition at line 262 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::track_access_mode().

```
00262                          {
00263     /* Interpret the shape which is a pointer to the first element as an
00264        array of Dimensions elements so that the range<Dimensions>
00265        constructor is happy with this collection
00266
00267        \todo Add also a constructor in range<> to accept a const
00268        std::size_t *?
00269     */
00270     return range<Dimensions> {
00271       *(const std::size_t (*)[Dimensions])(access.shape())
00272         };
00273   }
```

Here is the caller graph for this function:

8.1.2.7.3.8 get_size()

```
template<typename T, int Dimensions = 1>
auto cl::sycl::detail::buffer< T, Dimensions >::get_size ( ) const  [inline]
```

Returns the size of the buffer storage in bytes.

Todo rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf it is named bytes() for example

Definition at line 291 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::get_count().

```
00291                                {
00292     return get_count()*sizeof(value_type);
00293   }
```

Here is the call graph for this function:

8.1.2.7.3.9 mark_as_written()

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::mark_as_written ( )  [inline]
```

Enforce the buffer to be considered as being modified.

Same as creating an accessor with write access.

Definition at line 214 of file buffer.hpp.

References cl::sycl::access::host_buffer.

```
00214                                {
00215     modified = true;
00216   }
```

8.1.2.7.3.10 set_final_data() [1/4]

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
            std::weak_ptr< T > && final_data )  [inline]
```

Set the weak pointer as destination for write-back on buffer destruction.

Definition at line 299 of file buffer.hpp.

```
00299                                                        {
00300     final_write_back = [=] {
00301       if (auto sptr = final_data.lock()) {
00302         std::copy_n(access.data(), access.num_elements(), sptr.get());
00303       }
00304     };
00305   }
```

8.1.2.7.3.11 set_final_data() [2/4]

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
            std::shared_ptr< T > && final_data )  [inline]
```

Provide destination for write-back on buffer destruction as a shared pointer.

Definition at line 311 of file buffer.hpp.

```
00311                                                          {
00312     final_write_back = [=] {
00313       std::copy_n(access.data(), access.num_elements(), final_data.get());
00314     };
00315   }
```

8.1.2.7.3.12 set_final_data() [3/4]

```
template<typename T, int Dimensions = 1>
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
            std::nullptr_t  )  [inline]
```

Disable write-back on buffer destruction as an iterator.

Definition at line 320 of file buffer.hpp.

```
00320                                   {
00321     final_write_back = boost::none;
00322   }
```

8.1.2.7.3.13 set_final_data() [4/4]

```
template<typename T, int Dimensions = 1>
template<typename Iterator >
void cl::sycl::detail::buffer< T, Dimensions >::set_final_data (
            Iterator final_data )  [inline]
```

Provide destination for write-back on buffer destruction as an iterator.

Definition at line 329 of file buffer.hpp.

```
00329                                           {
00330  /*   using type_ = typename iterator_value_type<Iterator>::value_type;
00331     static_assert(std::is_same<type_, T>::value, "buffer type mismatch");
00332     static_assert(!(std::is_const<type_>::value),
00333                   "const iterator is not allowed");*/
00334     final_write_back = [=] {
00335       std::copy_n(access.data(), access.num_elements(), final_data);
00336     };
00337   }
```

8.1.2.7.3.14 track_access_mode()

```
template<typename T, int Dimensions = 1>
template<access::mode Mode, access::target Target = access::target::host_buffer>
void cl::sycl::detail::buffer< T, Dimensions >::track_access_mode ( )  [inline]
```

This method is to be called whenever an accessor is created.

Its current purpose is to track if an accessor with write access is created and acting accordingly.

Definition at line 226 of file buffer.hpp.

References cl::sycl::detail::buffer< T, Dimensions >::access, cl::sycl::detail::buffer< T, Dimensions >::allocate↩_buffer(), cl::sycl::access::atomic, cl::sycl::access::discard_read_write, cl::sycl::access::discard_write, cl::sycl↩::detail::buffer< T, Dimensions >::get_range(), cl::sycl::access::read_write, and cl::sycl::access::write.

```
                                {
    // test if write access is required
    if (   Mode == access::mode::write
        || Mode == access::mode::read_write
        || Mode == access::mode::discard_write
        || Mode == access::mode::discard_read_write
        || Mode == access::mode::atomic
       ) {
      modified = true;
      if (copy_if_modified) {
        // Implement the allocate & copy-on-write optimization
        copy_if_modified = false;
        data_host = false;
        // Since \c allocate_buffer() changes \c access, keep a copy first
        auto current_access = access;
        /* The range is actually computed from \c access itself, so
           save it */
        auto current_range = get_range();
        allocate_buffer(current_range);
        /* Then move everything to the new place

           \todo Use std::uninitialized_move instead, when we switch
           to full C++17
        */
        std::copy(current_access.begin(),
                  current_access.end(),
                  access.begin());
      }
    }
  }
```

Here is the call graph for this function:

8.1.2.7.4 Friends And Related Function Documentation

8.1.2.7.4.1 detail::accessor

```
template<typename T, int Dimensions = 1>
template<typename U , int D, access::mode Mode, access::target Target>
friend class detail::accessor  [friend]
```

Definition at line 63 of file buffer.hpp.

8.1.2.7.5 Member Data Documentation

8.1.2.7.5.1 access

```
template<typename T, int Dimensions = 1>
boost::multi_array_ref<value_type, Dimensions> cl::sycl::detail::buffer< T, Dimensions >↩
::access [private]
```

This is the multi-dimensional interface to the data that may point to either allocation in the case of storage managed by SYCL itself or to some other memory location in the case of host memory or storage<> abstraction use.

Definition at line 76 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::track_access_mode().

8.1.2.7.5.2 alloc

```
template<typename T, int Dimensions = 1>
std::allocator<non_const_value_type> cl::sycl::detail::buffer< T, Dimensions >::alloc [private]
```

The allocator to be used when some memory is needed.

Todo Implement user-provided allocator

Definition at line 69 of file buffer.hpp.

8.1.2.7.5.3 allocation

```
template<typename T, int Dimensions = 1>
non_const_value_type* cl::sycl::detail::buffer< T, Dimensions >::allocation = nullptr [private]
```

If some allocation is requested on the host for the buffer memory, this is where the memory is attached to.

Note that this is uninitialized memory, as stated in SYCL specification.

Definition at line 84 of file buffer.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer().

8.1.2.7.5.4 copy_if_modified

```
template<typename T, int Dimensions = 1>
bool cl::sycl::detail::buffer< T, Dimensions >::copy_if_modified = false [private]
```

Definition at line 99 of file buffer.hpp.

8.1.2.7.5.5 data_host

```
template<typename T, int Dimensions = 1>
bool cl::sycl::detail::buffer< T, Dimensions >::data_host = false  [private]
```

Definition at line 96 of file buffer.hpp.

8.1.2.7.5.6 final_write_back

```
template<typename T, int Dimensions = 1>
boost::optional<std::function<void(void)> > cl::sycl::detail::buffer< T, Dimensions >↩
::final_write_back  [private]
```

Definition at line 89 of file buffer.hpp.

8.1.2.7.5.7 input_shared_pointer

```
template<typename T, int Dimensions = 1>
shared_ptr_class<T> cl::sycl::detail::buffer< T, Dimensions >::input_shared_pointer  [private]
```

Definition at line 92 of file buffer.hpp.

8.1.2.7.5.8 modified

```
template<typename T, int Dimensions = 1>
bool cl::sycl::detail::buffer< T, Dimensions >::modified = false  [private]
```

Definition at line 102 of file buffer.hpp.

8.1.2.8 struct cl::sycl::detail::buffer_base

Factorize some template independent buffer aspects in a base class.

Definition at line 50 of file buffer_base.hpp.

Inheritance diagram for cl::sycl::detail::buffer_base:

Collaboration diagram for cl::sycl::detail::buffer_base:

Public Member Functions

- buffer_base ()

  Create a buffer base and marks the host context as the context that holds the most recent version of the data.
- ~buffer_base ()

  The destructor wait for not being used anymore.
- void wait ()

  Wait for this buffer to be ready, which is no longer in use.
- void use ()

  Mark this buffer in use by a task.
- void release ()

  A task has released the buffer.
- std::shared_ptr< detail::task > get_latest_producer ()

  Return the latest producer for the buffer.
- std::shared_ptr< detail::task > set_latest_producer (std::weak_ptr< detail::task > newer_latest_producer)

  Return the latest producer for the buffer and set another future producer.
- std::shared_ptr< detail::task > add_to_task (handler *command_group_handler, bool is_write_mode)

  Add a buffer to the task running the command group.
- bool is_data_up_to_date (const cl::sycl::context &ctx)

  Check if the data of this buffer is up-to-date in a certain context.
- bool is_cached (const cl::sycl::context &ctx)

  Check if the buffer is already cached for a certain context.
- void create_in_cache (const cl::sycl::context &ctx, size_t size, cl_mem_flags flags, void *data)

Create a `boost::compute::buffer` for this `cl::sycl::buffer` in the cache and associate it with a given context.

- void sync_with_host (std::size_t size, void ∗data)

  Transfer the most up-to-date version of the data to the host if the host version is not already up-to-date.
- void update_buffer_state (const cl::sycl::context &target_ctx, access::mode mode, std::size_t size, void ∗data)

  When a transfer is requested this function is called, it will update the state of the buffer according to the context in which the accessor is created and the access mode.
- boost::compute::buffer get_cl_buffer (const cl::sycl::context &context)

  Returns the cl_buffer for a given context.

Public Attributes

- std::atomic< size_t > number_of_users
- std::weak_ptr< detail::task > latest_producer

  Track the latest task to produce this buffer.
- std::mutex latest_producer_mutex

  To protect the access to latest_producer.
- std::condition_variable ready

  To signal when this buffer ready.
- std::mutex ready_mutex

  To protect the access to the condition variable.
- boost::optional< std::promise< void > > notify_buffer_destructor

  If the SYCL user buffer destructor is blocking, use this to block until this buffer implementation is destroyed.
- std::unordered_set< cl::sycl::context > fresh_ctx

  To track contexts in which the data is up-to-date.
- std::unordered_map< cl::sycl::context, boost::compute::buffer > buffer_cache

  Buffer-side cache that keeps the `boost::compute::buffer` (and the underlying `cl_buffer`) so that if the buffer already exists inside the same context it is not recreated.

8.1.2.8.1 Constructor & Destructor Documentation

8.1.2.8.1.1 buffer_base()

```
cl::sycl::detail::buffer_base::buffer_base ( )  [inline]
```

Create a buffer base and marks the host context as the context that holds the most recent version of the data.

Todo Use lazy allocation for the context tracking set

Definition at line 87 of file buffer_base.hpp.

```
00087                 : number_of_users { 0 },
00088                   fresh_ctx { cl::sycl::context {} } {}
```

8.1.2.8.1.2 ~buffer_base()

```
cl::sycl::detail::buffer_base::~buffer_base ( ) [inline]
```

The destructor wait for not being used anymore.

Definition at line 92 of file buffer_base.hpp.

References wait().

```
00092                 {
00093     wait();
00094     // If there is the last SYCL user buffer waiting, notify it
00095     if (notify_buffer_destructor)
00096       notify_buffer_destructor->set_value();
00097   }
```

Here is the call graph for this function:

8.1.2.8.2 Member Function Documentation

8.1.2.8.2.1 add_to_task()

```
std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::add_to_task (
            handler * command_group_handler,
            bool is_write_mode ) [inline]
```

Add a buffer to the task running the command group.

Definition at line 149 of file buffer_base.hpp.

References cl::sycl::detail::add_buffer_to_task().

```
00149                                                                   {
00150     return add_buffer_to_task(command_group_handler,
00151                               shared_from_this(),
00152                               is_write_mode);
00153   }
```

Here is the call graph for this function:

8.1.2.8.2.2 create_in_cache()

```
void cl::sycl::detail::buffer_base::create_in_cache (
            const cl::sycl::context & ctx,
            size_t size,
            cl_mem_flags flags,
            void * data )  [inline]
```

Create a `boost::compute::buffer` for this `cl::sycl::buffer` in the cache and associate it with a given context.

Definition at line 172 of file buffer_base.hpp.

References cl::sycl::context::get_boost_compute().

Referenced by update_buffer_state().

```
00173                                                   {
00174     buffer_cache[ctx] = boost::compute::buffer
00175       { ctx.get_boost_compute(),
00176         size,
00177         flags,
00178         data
00179       };
00180   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.8.2.3 get_cl_buffer()

```
boost::compute::buffer cl::sycl::detail::buffer_base::get_cl_buffer (
            const cl::sycl::context & context )  [inline]
```

Returns the cl_buffer for a given context.

Definition at line 308 of file buffer_base.hpp.

```
00308                                                   {
00309     return buffer_cache[context];
00310   }
```

8.1.2.8.2.4 get_latest_producer()

```
std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::get_latest_producer ( ) [inline]
```

Return the latest producer for the buffer.

Definition at line 126 of file buffer_base.hpp.

```
00126                                                   {
00127     std::lock_guard<std::mutex> lg { latest_producer_mutex };
00128     // Return the valid shared_ptr to the task, if any
00129     return latest_producer.lock();
00130   }
```

8.1.2.8.2.5 is_cached()

```
bool cl::sycl::detail::buffer_base::is_cached (
            const cl::sycl::context & ctx ) [inline]
```

Check if the buffer is already cached for a certain context.

Definition at line 164 of file buffer_base.hpp.

Referenced by update_buffer_state().

```
00164                                           {
00165     return buffer_cache.count(ctx);
00166   }
```

Here is the caller graph for this function:

8.1.2.8.2.6 is_data_up_to_date()

```
bool cl::sycl::detail::buffer_base::is_data_up_to_date (
            const cl::sycl::context & ctx ) [inline]
```

Check if the data of this buffer is up-to-date in a certain context.

Definition at line 158 of file buffer_base.hpp.

Referenced by sync_with_host(), and update_buffer_state().

```
00158                                                   {
00159     return fresh_ctx.count(ctx);
00160   }
```

Here is the caller graph for this function:

8.1.2.8.2.7 release()

```
void cl::sycl::detail::buffer_base::release ( )  [inline]
```

A task has released the buffer.

Definition at line 118 of file buffer_base.hpp.

```
00118                 {
00119     if (--number_of_users == 0)
00120       // Notify the host consumers or the buffer destructor that it is ready
00121       ready.notify_all();
00122   }
```

8.1.2.8.2.8 set_latest_producer()

```
std::shared_ptr<detail::task> cl::sycl::detail::buffer_base::set_latest_producer (
            std::weak_ptr< detail::task > newer_latest_producer )  [inline]
```

Return the latest producer for the buffer and set another future producer.

Definition at line 137 of file buffer_base.hpp.

```
00137                                                                 {
00138     std::lock_guard<std::mutex> lg { latest_producer_mutex };
00139     using std::swap;
00140
00141     swap(newer_latest_producer, latest_producer);
00142     // Return the valid shared_ptr to the previous producing task, if any
00143     return newer_latest_producer.lock();
00144   }
```

8.1.2.8.2.9 sync_with_host()

```
void cl::sycl::detail::buffer_base::sync_with_host (
            std::size_t size,
            void * data )  [inline]
```

Transfer the most up-to-date version of the data to the host if the host version is not already up-to-date.

Definition at line 186 of file buffer_base.hpp.

References cl::sycl::detail::host_context::get_boost_queue(), and is_data_up_to_date().

Referenced by update_buffer_state().

```
00186                                                 {
00187     cl::sycl::context host_context;
00188     if (!is_data_up_to_date(host_context) && !fresh_ctx.empty()) {
00189       /* We know that the context(s) in \c fresh_ctx hold the most recent
00190          version of the buffer
00191       */
00192       auto fresh_context = *(fresh_ctx.begin());
00193       auto fresh_q = fresh_context.get_boost_queue();
00194       fresh_q.enqueue_read_buffer(buffer_cache[fresh_context], 0, size, data);
00195       fresh_ctx.insert(host_context);
00196     }
00197   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.8.2.10 update_buffer_state()

```
void cl::sycl::detail::buffer_base::update_buffer_state (
            const cl::sycl::context & target_ctx,
            access::mode mode,
            std::size_t size,
            void * data )  [inline]
```

When a transfer is requested this function is called, it will update the state of the buffer according to the context in which the accessor is created and the access mode.

Definition at line 204 of file buffer_base.hpp.

References cl::sycl::access::atomic, create_in_cache(), cl::sycl::access::discard_read_write, cl::sycl::access↩ ::discard_write, cl::sycl::context::get_boost_queue(), is_cached(), is_data_up_to_date(), cl::sycl::context::is_host(), cl::sycl::access::read, cl::sycl::access::read_write, sync_with_host(), and cl::sycl::access::write.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::call_update_buffer_state().

```
00205                                                                   {
00206     /* The \c cl_buffer we put in the cache might get accessed again in the
00207        future, this means that we have to always to create it in read/write
00208        mode to be able to write to it if it is accessed through a
00209        write accessor in the future
00210      */
00211     auto constexpr flag = CL_MEM_READ_WRITE;
00212
00213     /* The buffer is accessed in read mode, we want to transfer the data only if
00214        necessary. We start a transfer if the data on the target context is not
00215        up to date and then update the fresh context set.
00216      */
00217     if (mode == access::mode::read) {
```

```

      if (is_data_up_to_date(target_ctx))
        // If read mode and the data is up-to-date there is nothing to do
        return;

      // The data is not up-to-date, we need a transfer
      // We also want to be sure that the host holds the most recent data
      sync_with_host(size, data);

      if (!target_ctx.is_host()) {
        // If the target context is a device context
        if (!is_cached(target_ctx)) {
          /* If not cached, we create the buffer and copy the data
             at the same time
          */
          create_in_cache(target_ctx, size,
                          (flag | CL_MEM_COPY_HOST_PTR), data);
          fresh_ctx.insert(target_ctx);
          return;
        }

        /* Else we transfer the data to the existing buffer associated
           with the target context buffer
        */
        auto q = target_ctx.get_boost_queue();
        q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
        fresh_ctx.insert(target_ctx);
      }
      return;
    }

    /* The buffer might be written to, this means that we have to consider
       every version of the data obsolete except in the target context

       We go through the same process as in read mode but in addition
       we empty the fresh context set and just add the target context

       If the data is up to date on the target we just have to update
       the context set and nothing else
    */
    if (!is_data_up_to_date(target_ctx)) {

      if (   mode == access::mode::read_write
          || mode == access::mode::write
          || mode == access::mode::atomic) {
        // If the data is not up-to-date in the target context
        // We want to host to be up-to-date
        sync_with_host(size, data);

        if (!target_ctx.is_host()) {
          // If the target context is a device context
          if (!is_cached(target_ctx)) {
            create_in_cache(target_ctx, size,
                            (flag | CL_MEM_COPY_HOST_PTR), data);
          }
          else {
            // We update the buffer associated with the target context
            auto q = target_ctx.get_boost_queue();
            q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
          }
        }
      }

      /* When in discard mode we don't need to transfer any data, we just create
         the \c cl_buffer if it doesn't exist in the cache
      */
      if (   mode == access::mode::discard_write
          || mode == access::mode::discard_read_write) {
        /* We only need to create the buffer if it doesn't exist
           but without copying any data because of the discard mode
        */
        if (!target_ctx.is_host() && !is_cached(target_ctx)) {
          // If the context doesn't exist we create it.
          /* We don't want to transfer any data so we don't
             add \c CL_MEM_COPY_HOST_PTR
          */
          create_in_cache(target_ctx, size, flag, 0);
        }
      }
    }
    /* Here we are sure that we are in some kind of write mode,
       we indicate that all contexts except the target context
       are not up-to-date anymore
    */
    fresh_ctx.clear();
    fresh_ctx.insert(target_ctx);
  }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.8.2.11 use()

```
void cl::sycl::detail::buffer_base::use ( ) [inline]
```

Mark this buffer in use by a task.

Definition at line 111 of file buffer_base.hpp.

References number_of_users.

```
00111             {
00112     // Increment the use count
00113     ++number_of_users;
00114   }
```

8.1.2.8.2.12 wait()

```
void cl::sycl::detail::buffer_base::wait ( ) [inline]
```

Wait for this buffer to be ready, which is no longer in use.

Definition at line 101 of file buffer_base.hpp.

Referenced by ~buffer_base().

```
00101               {
00102     std::unique_lock<std::mutex> ul { ready_mutex };
00103     ready.wait(ul, [&] {
00104         // When there is no producer for this buffer, we are ready to use it
00105         return number_of_users == 0;
00106       });
00107   }
```

Here is the caller graph for this function:

8.1.2.8.3 Member Data Documentation

8.1.2.8.3.1 buffer_cache

```
std::unordered_map<cl::sycl::context, boost::compute::buffer> cl::sycl::detail::buffer_base↩
::buffer_cache
```

Buffer-side cache that keeps the `boost::compute::buffer` (and the underlying `cl_buffer`) so that if the buffer already exists inside the same context it is not recreated.

Definition at line 80 of file buffer_base.hpp.

8.1.2.8.3.2 fresh_ctx

```
std::unordered_set<cl::sycl::context> cl::sycl::detail::buffer_base::fresh_ctx
```

To track contexts in which the data is up-to-date.

Definition at line 73 of file buffer_base.hpp.

8.1.2.8.3.3 latest_producer

```
std::weak_ptr<detail::task> cl::sycl::detail::buffer_base::latest_producer
```

Track the latest task to produce this buffer.

Definition at line 56 of file buffer_base.hpp.

8.1.2.8.3.4 latest_producer_mutex

```
std::mutex cl::sycl::detail::buffer_base::latest_producer_mutex
```

To protect the access to latest_producer.

Definition at line 58 of file buffer_base.hpp.

8.1.2.8.3.5 notify_buffer_destructor

```
boost::optional<std::promise<void> > cl::sycl::detail::buffer_base::notify_buffer_destructor
```

If the SYCL user buffer destructor is blocking, use this to block until this buffer implementation is destroyed.

Use a void promise since there is no value to send, only waiting

Definition at line 70 of file buffer_base.hpp.

Referenced by cl::sycl::detail::buffer< T, Dimensions >::get_destructor_future().

8.1.2.8.3.6 number_of_users

```
std::atomic<size_t> cl::sycl::detail::buffer_base::number_of_users
```

Definition at line 53 of file buffer_base.hpp.

Referenced by use().

8.1.2.8.3.7 ready

```
std::condition_variable cl::sycl::detail::buffer_base::ready
```

To signal when this buffer ready.

Definition at line 61 of file buffer_base.hpp.

8.1.2.8.3.8 ready_mutex

```
std::mutex cl::sycl::detail::buffer_base::ready_mutex
```

To protect the access to the condition variable.

Definition at line 63 of file buffer_base.hpp.

8.1.2.9 class cl::sycl::detail::buffer_waiter

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>
class cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >

A helper class to wait for the final buffer destruction if the conditions for blocking are met.

Definition at line 33 of file buffer_waiter.hpp.

Inheritance diagram for cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >:

Collaboration diagram for cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >:

Public Member Functions

- buffer_waiter (detail::buffer< T, Dimensions > *b)

  Create a new buffer_waiter on top of a detail::buffer.
- ~buffer_waiter ()

  The buffer_waiter destructor waits for any data to be written back to the host, if any.

Private Types

- using implementation_t = typename buffer_waiter::shared_ptr_implementation

Private Attributes

- friend implementation_t

Additional Inherited Members

8.1.2.9.1 Member Typedef Documentation

8.1.2.9.1.1 implementation_t

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
using cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::implementation_t = typename
buffer_waiter::shared_ptr_implementation  [private]
```

Definition at line 41 of file buffer_waiter.hpp.

8.1.2.9.2 Constructor & Destructor Documentation

8.1.2.9.2.1 buffer_waiter()

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::buffer_waiter (
            detail::buffer< T, Dimensions > * b )  [inline]
```

Create a new buffer_waiter on top of a detail::buffer.

Definition at line 52 of file buffer_waiter.hpp.

```
00052 : implementation_t { b } {}
```

8.1.2.9.2.2 ~buffer_waiter()

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::~buffer_waiter ( )  [inline]
```

The buffer_waiter destructor waits for any data to be written back to the host, if any.

Definition at line 58 of file buffer_waiter.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer_waiter< T, Dimensions, Allocator >, detail::buffer< T, Dimensions > >::implementation, and TRISYCL_DUMP_T.

```
00058                    {
00059     /* Get a future from the implementation if we have to wait for its
00060        destruction */
00061     auto f = implementation->get_destructor_future();
00062     if (f) {
00063       /* No longer carry for the implementation buffer which is free to
00064          live its life up to its destruction */
00065       implementation.reset();
00066       TRISYCL_DUMP_T("~buffer_waiter() is waiting");
00067       // Then wait for its end in some other thread
00068       f->wait();
00069       TRISYCL_DUMP_T("~buffer_waiter() is done");
00070     }
00071   }
```

8.1.2.9.3 Member Data Documentation

8.1.2.9.3.1 implementation_t

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
friend cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::implementation_t  [private]
```

Definition at line 44 of file buffer_waiter.hpp.

8.1.2.10 class cl::sycl::buffer

template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>
class cl::sycl::buffer< T, Dimensions, Allocator >

A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on.

Todo There is a naming inconsistency in the specification between buffer and accessor on T versus datatype

Todo Finish allocator implementation

Todo Think about the need of an allocator when constructing a buffer from other buffers

Todo Update the specification to have a non-const allocator for const buffer? Or do we rely on rebind_alloc<T>. But does this work with astate-full allocator?

Todo Add constructors from arrays so that in C++17 the range and type can be infered from the constructor

Todo Add constructors from array_ref

Definition at line 29 of file accessor.hpp.

Inheritance diagram for cl::sycl::buffer< T, Dimensions, Allocator >:

Collaboration diagram for cl::sycl::buffer< T, Dimensions, Allocator >:

Public Types

- using value_type = T

  The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &
- using allocator_type = Allocator

Public Member Functions

- buffer ()=default

  Use default constructors so that we can create a new buffer copy from another one, with either a l-value or an r-value (for std::move() for example).
- buffer (const range< Dimensions > &r, Allocator allocator={})

  Create a new buffer of the given size with storage managed by the SYCL runtime.
- template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>::value>>
  buffer (const T *host_data, const range< Dimensions > &r, Allocator allocator={})

  Create a new buffer with associated host memory.
- buffer (T *host_data, const range< Dimensions > &r, Allocator allocator={})

  Create a new buffer with associated host memory.
- buffer (shared_ptr_class< T > &host_data, const range< Dimensions > &buffer_range, cl::sycl::mutex_class &m, Allocator allocator={})

  Create a new buffer with associated memory, using the data in host_data.
- buffer (shared_ptr_class< T > host_data, const range< Dimensions > &buffer_range, Allocator allocator={})

  Create a new buffer with associated memory, using the data in host_data.
- template<typename InputIterator , typename ValueType = typename std::iterator_traits<InputIterator>::value_type>
  buffer (InputIterator start_iterator, InputIterator end_iterator, Allocator allocator={})

  Create a new allocated 1D buffer initialized from the given elements ranging from first up to one before last.
- buffer (buffer< T, Dimensions, Allocator > &b, const id< Dimensions > &base_index, const range< Dimensions > &sub_range, Allocator allocator={})

  Create a new sub-buffer without allocation to have separate accessors later.
- buffer (cl_mem mem_object, queue from_queue, event available_event={}, Allocator allocator={})

  Create a buffer from an existing OpenCL memory object associated with a context after waiting for an event signaling the availability of the OpenCL data.
- template<access::mode Mode, access::target Target = access::target::global_buffer>
  accessor< T, Dimensions, Mode, Target > get_access (handler &command_group_handler)

  Get an accessor to the buffer with the required mode.
- void mark_as_written ()

  Force the buffer to behave like if we had created an accessor in write mode.
- template<access::mode Mode, access::target Target = access::target::host_buffer>
  accessor< T, Dimensions, Mode, Target > get_access ()

  Get a host accessor to the buffer with the required mode.
- auto get_range () const

  Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.
- auto get_count () const

  Returns the total number of elements in the buffer.
- size_t get_size () const

  Returns the size of the buffer storage in bytes.
- auto use_count () const

  Returns the number of buffers that are shared/referenced.
- bool constexpr is_read_only () const

  Ask for read-only status of the buffer.
- void set_final_data (shared_ptr_class< T > finalData)

  Set destination of buffer data on destruction.
- void set_final_data (weak_ptr_class< T > finalData)

  Set destination of buffer data on destruction.
- void set_final_data (std::nullptr_t)

  Disable write-back on buffer destruction.
- bool is_cached (cl::sycl::context &ctx)

Check if the buffer is already cached in a certain context.

- bool is_data_up_to_date (cl::sycl::context &ctx)

  Check if the data stored in the buffer is up-to-date in a certain context.

- template<typename Iterator >
  void set_final_data (Iterator &&finalData)

  Set destination of buffer data on destruction.

Private Types

- using implementation_t = typename buffer::shared_ptr_implementation

Private Attributes

- friend implementation_t

Additional Inherited Members

8.1.2.10.1 Member Typedef Documentation

8.1.2.10.1.1 allocator_type

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::allocator_type = Allocator
```

Definition at line 75 of file buffer.hpp.

8.1.2.10.1.2 const_reference

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::const_reference = const value_type&
```

Definition at line 74 of file buffer.hpp.

8.1.2.10.1.3 implementation_t

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::implementation_t = typename buffer::shared↩
_ptr_implementation  [private]
```

Definition at line 80 of file buffer.hpp.

8.1.2.10.1.4 reference

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::reference = value_type&
```

Definition at line 73 of file buffer.hpp.

8.1.2.10.1.5 value_type

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
using cl::sycl::buffer< T, Dimensions, Allocator >::value_type = T
```

The STL-like types.

Definition at line 72 of file buffer.hpp.

8.1.2.10.2 Constructor & Destructor Documentation

8.1.2.10.2.1 buffer() [1/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer ( )  [default]
```

Use default constructors so that we can create a new buffer copy from another one, with either a l-value or an r-value (for std::move() for example).

Since we just copy the shared_ptr<> from the shared_ptr_implementation above, this is where/how the sharing magic is happening with reference counting in this case.

8.1.2.10.2.2 buffer() [2/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            const range< Dimensions > & r,
            Allocator allocator = {} )  [inline]
```

Create a new buffer of the given size with storage managed by the SYCL runtime.

The default behavior is to use the default host buffer allocator, in order to allow for host accesses. If the type of the buffer, has the const qualifier, then the default allocator will remove the qualifier to allow host access to the data.

Parameters

in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime

Definition at line 113 of file buffer.hpp.

```
00113                                                                {})
00114     : implementation_t { detail::waiter<T, Dimensions, Allocator>(
00115                          new detail::buffer<T, Dimensions> { r }) }
00116       {}
```

8.1.2.10.2.3 buffer() [3/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
template<typename Dependent = T, typename = std::enable_if_t<!std::is_const<Dependent>↩
::value>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            const T * host_data,
            const range< Dimensions > & r,
            Allocator allocator = {} )  [inline]
```

Create a new buffer with associated host memory.

Parameters

in	*host_data*	points to the storage and values used by the buffer
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime, of type `cl::sycl::buffer_allocator<T>` by default

The host address is

```
const T*
```

, so the host memory is read-only.

However, the typename T is not const so the device accesses can be both read and write accesses. Since, the host_data is const, this buffer is only initialized with this memory and there is no write after its destruction, unless there is another final data address given after construction of the buffer.

Only enable this constructor if it is not the same as the one with

```
const T *host_data
```

, which is when `T` is already a constant type.

Todo Actually this is redundant.

Definition at line 146 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                                {})
    : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
                          { host_data, r }) }
  {}
```

Here is the call graph for this function:

8.1.2.10.2.4 buffer() [4/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            T * host_data,
            const range< Dimensions > & r,
            Allocator allocator = {} )  [inline]
```

Create a new buffer with associated host memory.

Parameters

in,out	*host_data*	points to the storage and values used by the buffer
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The memory is owned by the runtime during the lifetime of the object. Data is copied back to the host unless the user overrides the behavior using the set_final_data method. host_data points to the storage and values used by the buffer and range<Dimensions> defines the size.

Definition at line 170 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
                                {})
    : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
                          { host_data, r }) }
  {}
```

Here is the call graph for this function:

8.1.2.10.2.5 buffer() [5/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            shared_ptr_class< T > & host_data,
            const range< Dimensions > & buffer_range,
            cl::sycl::mutex_class & m,
            Allocator allocator = {} )  [inline]
```

Create a new buffer with associated memory, using the data in host_data.

Parameters

in,out	*host_data*	points to the storage and values used by the buffer
in	*r*	defines the size
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The ownership of the host_data is shared between the runtime and the user. In order to enable both the user application and the SYCL runtime to use the same pointer, a cl::sycl::mutex_class is used. The mutex m is locked by the runtime whenever the data is in use and unlocked otherwise. Data is synchronized with host_data, when the mutex is unlocked by the runtime.

Todo update the specification to replace the pointer by a reference and provide the constructor with and without a mutex

Definition at line 199 of file buffer.hpp.

References cl::sycl::detail::unimplemented().

```
00202                                {}) {
00203     detail::unimplemented();
00204   }
```

Here is the call graph for this function:

8.1.2.10.2.6 buffer() [6/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            shared_ptr_class< T > host_data,
            const range< Dimensions > & buffer_range,
            Allocator allocator = {} )  [inline]
```

Create a new buffer with associated memory, using the data in host_data.

Parameters

in,out	*host_data*	points to the storage and values used by the buffer
in	*r*	defines the size
in,out	*m*	is the mutex used to protect the data access
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

The ownership of the host_data is shared between the runtime and the user. In order to enable both the user application and the SYCL runtime to use the same pointer, a cl::sycl::mutex_class is used.

Todo add this mutex-less constructor to the specification

Definition at line 227 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
00229                                {})
00230     : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00231                          { host_data, buffer_range }) }
00232   {}
```

Here is the call graph for this function:

8.1.2.10.2.7 buffer() [7/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
template<typename InputIterator , typename ValueType = typename std::iterator_traits<Input↩
Iterator>::value_type>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            InputIterator start_iterator,
            InputIterator end_iterator,
            Allocator allocator = {} )  [inline]
```

Create a new allocated 1D buffer initialized from the given elements ranging from first up to one before last.

The data is copied to an intermediate memory position by the runtime. Data is written back to the same iterator set if the iterator is not a const iterator.

Parameters

in,out	*start_iterator*	points to the first element to copy
in	*end_iterator*	points to just after the last element to copy
in	*allocator*	is to be used by the SYCL runtime, of type cl::sycl::buffer_allocator<T> by default

Todo Implement the copy back at buffer destruction

Todo Generalize this for n-D and provide column-major and row-major initialization

Todo a reason to have this nD is that set_final_data(weak_ptr_class<T> & finalData) is actually doing this linearization anyway

Todo Allow read-only buffer construction too

Todo update the specification to deal with forward iterators instead and rewrite back only when it is non const and output iterator at least

Todo Allow initialization from ranges and collections à la STL

Definition at line 272 of file buffer.hpp.

References cl::sycl::detail::waiter().

```
00274                                {}) :
00275     implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
00276                        { start_iterator, end_iterator }) }
00277   {}
```

Here is the call graph for this function:

8.1.2.10.2.8 buffer() [8/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            buffer< T, Dimensions, Allocator > & b,
            const id< Dimensions > & base_index,
            const range< Dimensions > & sub_range,
            Allocator allocator = {} )  [inline]
```

Create a new sub-buffer without allocation to have separate accessors later.

Parameters

in,out	*b*	is the buffer with the real data
in	*base_index*	specifies the origin of the sub-buffer inside the buffer b
in	*sub_range*	specifies the size of the sub-buffer

Todo To be implemented

Todo Update the specification to replace index by id

Definition at line 294 of file buffer.hpp.

References cl::sycl::detail::unimplemented().

```
00297                                {}) { detail::unimplemented(); }
```

Here is the call graph for this function:

8.1.2.10.2.9 buffer() [9/9]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
cl::sycl::buffer< T, Dimensions, Allocator >::buffer (
            cl_mem mem_object,
            queue from_queue,
            event available_event = {},
            Allocator allocator = {} )  [inline]
```

Create a buffer from an existing OpenCL memory object associated with a context after waiting for an event signaling the availability of the OpenCL data.

Parameters

in,out	*mem_object*	is the OpenCL memory object to use
in,out	*from_queue*	is the queue associated to the memory object
in	*available_event*	specifies the event to wait for if non null

Note that a buffer created from a cl_mem object will only have one underlying cl_mem for the lifetime of the buffer and use on an incompatible queue constitues an error.

Todo To be implemented

Todo Improve the specification to allow CLHPP objects too

Definition at line 321 of file buffer.hpp.

References cl::sycl::access::global_buffer, and cl::sycl::detail::unimplemented().

```
00323                                   {},
00324           Allocator allocator = {}) { detail::unimplemented();  }
```

Here is the call graph for this function:

8.1.2.10.3 Member Function Documentation

8.1.2.10.3.1 get_access() [1/2]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
template<access::mode Mode, access::target Target = access::target::global_buffer>
accessor<T, Dimensions, Mode, Target> cl::sycl::buffer< T, Dimensions, Allocator >::get_↩
access (
            handler & command_group_handler )  [inline]
```

Get an accessor to the buffer with the required mode.

Parameters

	Mode	is the requested access mode
	Target	is the type of object to be accessed
in	*command_group_handler*	is the command group handler in which the kernel is to be executed

Todo Do we need for an accessor to increase the reference count of a buffer object? It does make more sense for a host-side accessor.

Todo Implement the modes and targets

Definition at line 347 of file buffer.hpp.

References cl::sycl::access::constant_buffer, cl::sycl::access::global_buffer, and cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00347                                                {
00348     static_assert(Target == access::target::global_buffer
00349                   || Target == access::target::constant_buffer,
00350                   "get_access(handler) can only deal with access::global_buffer"
00351                   " or access::constant_buffer (for host_buffer accessor"
00352                   " do not use a command group handler");
00353     implementation->implementation->template track_access_mode<Mode, Target>();
00354     return { *this, command_group_handler };
00355   }
```

8.1.2.10.3.2 get_access() [2/2]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
template<access::mode Mode, access::target Target = access::target::host_buffer>
accessor<T, Dimensions, Mode, Target> cl::sycl::buffer< T, Dimensions, Allocator >::get_↩
access ( )  [inline]
```

Get a host accessor to the buffer with the required mode.

Parameters

Mode	is the requested access mode

Todo Implement the modes

Todo More elegant solution

Definition at line 377 of file buffer.hpp.

References cl::sycl::access::host_buffer, and cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00377                {
00378     static_assert(Target == access::target::host_buffer,
00379                   "get_access() without a command group handler is only"
00380                   " for host_buffer accessor");
00381     implementation->implementation->template track_access_mode<Mode, Target>();
00382     return { *this };
00383   }
```

8.1.2.10.3.3 get_count()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
auto cl::sycl::buffer< T, Dimensions, Allocator >::get_count ( ) const  [inline]
```

Returns the total number of elements in the buffer.

Equal to get_range()[0] ∗ ... ∗ get_range()[Dimensions-1].

Definition at line 407 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00407                        {
00408     return implementation->implementation->get_count();
00409   }
```

8.1.2.10.3.4 get_range()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
auto cl::sycl::buffer< T, Dimensions, Allocator >::get_range ( ) const  [inline]
```

Return a range object representing the size of the buffer in terms of number of elements in each dimension as passed to the constructor.

Todo rename to the equivalent from array_ref proposals? Such as size() in http://www.open-std.↩org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html

Definition at line 394 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00394                        {
00395     /* Interpret the shape which is a pointer to the first element as an
00396        array of Dimensions elements so that the range<Dimensions>
00397        constructor is happy with this collection
00398     */
00399     return implementation->implementation->get_range();
00400   }
```

8.1.2.10.3.5 get_size()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
size_t cl::sycl::buffer< T, Dimensions, Allocator >::get_size ( ) const  [inline]
```

Returns the size of the buffer storage in bytes.

Equal to get_count()∗sizeof(T).

Todo rename to something else. In http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf it is named bytes() for example

Definition at line 420 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00420                                {
00421     return implementation->implementation->get_size();
00422   }
```

8.1.2.10.3.6 is_cached()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
bool cl::sycl::buffer< T, Dimensions, Allocator >::is_cached (
            cl::sycl::context & ctx )  [inline]
```

Check if the buffer is already cached in a certain context.

Definition at line 501 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00501                                        {
00502     return implementation->implementation->is_cached(ctx);
00503   }
```

8.1.2.10.3.7 is_data_up_to_date()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
bool cl::sycl::buffer< T, Dimensions, Allocator >::is_data_up_to_date (
            cl::sycl::context & ctx )  [inline]
```

Check if the data stored in the buffer is up-to-date in a certain context.

Definition at line 508 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00508                                                {
00509     return implementation->implementation->is_data_up_to_date(ctx);
00510   }
```

8.1.2.10.3.8 is_read_only()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
bool constexpr cl::sycl::buffer< T, Dimensions, Allocator >::is_read_only ( ) const  [inline]
```

Ask for read-only status of the buffer.

Todo Add to specification

Definition at line 447 of file buffer.hpp.

```
00447                                   {
00448     return std::is_const<T>::value;
00449   }
```

8.1.2.10.3.9 mark_as_written()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::mark_as_written ( )  [inline]
```

Force the buffer to behave like if we had created an accessor in write mode.

Definition at line 361 of file buffer.hpp.

References cl::sycl::access::host_buffer, and cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00361                         {
00362     return implementation->implementation->mark_as_written();
00363   }
```

8.1.2.10.3.10 set_final_data() [1/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
            shared_ptr_class< T > finalData )  [inline]
```

Set destination of buffer data on destruction.

The finalData points to the host memory to which, the outcome of all the buffer processing is going to be copied to.

This is the final pointer, which is going to be accessible after the destruction of the buffer and in the case where this is a valid pointer, the data are going to be copied to this host address.

finalData is different from the original host address, if the buffer was created associated with one. This is mainly to be used when a shared_ptr is given in the constructor and the output data will reside in a different location from the initialization data.

It is defined as a weak_ptr referring to a shared_ptr that is not associated with the cl::sycl::buffer, and so the cl↩::sycl::buffer will have no ownership of finalData.

Todo Update the API to take finalData by value instead of by reference. This way we can have an implicit conversion possible at the API call from a shared_ptr<>, avoiding an explicit weak_ptr<> creation

Todo figure out how set_final_data() interact with the other way to write back some data or with some data sharing with the host that can not be undone

Definition at line 479 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

Referenced by cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data().

```
00479                                                {
00480     implementation->implementation->set_final_data(std::move(finalData));
00481   }
```

Here is the caller graph for this function:

8.1.2.10.3.11 set_final_data() [2/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
            weak_ptr_class< T > finalData )  [inline]
```

Set destination of buffer data on destruction.

Definition at line 486 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00486                                                {
00487     implementation->implementation->set_final_data(std::move(finalData));
00488   }
```

8.1.2.10.3.12 set_final_data() [3/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
            std::nullptr_t  )  [inline]
```

Disable write-back on buffer destruction.

Definition at line 493 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00493                                   {
00494     implementation->implementation->set_final_data(nullptr);
00495   }
```

8.1.2.10.3.13 set_final_data() [4/4]

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
template<typename Iterator >
void cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data (
            Iterator && finalData )  [inline]
```

Set destination of buffer data on destruction.

WARNING: the user has to ensure that the object refered to by the iterator will be alive after buffer destruction, otherwise the behaviour is undefined.

Definition at line 520 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation, and cl::sycl::buffer< T, Dimensions, Allocator >::set_final_data().

```
00520                                           {
00521     implementation->implementation->
00522       set_final_data(std::forward<Iterator>(finalData));
00523   }
```

Here is the call graph for this function:

8.1.2.10.3.14 use_count()

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
auto cl::sycl::buffer< T, Dimensions, Allocator >::use_count ( ) const [inline]
```

Returns the number of buffers that are shared/referenced.

For example

```
cl::sycl::buffer<int> b { 1000 };
// Here b.use_count() should return 1
cl::sycl::buffer<int> c { b };
// Here b.use_count() and b.use_count() should return 2
```

Todo Add to the specification, useful for validation

Definition at line 437 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::implementation.

```
00437                                {
00438     // Rely on the shared_ptr<> use_count()
00439     return implementation.use_count();
00440   }
```

8.1.2.10.4 Member Data Documentation

8.1.2.10.4.1 implementation_t

```
template<typename T, int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
friend cl::sycl::buffer< T, Dimensions, Allocator >::implementation_t [private]
```

Definition at line 83 of file buffer.hpp.

8.1.2.11 struct cl::sycl::image

template<int Dimensions>
struct cl::sycl::image< Dimensions >

Todo implement image

Definition at line 23 of file image.hpp.

8.1.2.12 struct cl::sycl::detail::reserve_id

template<typename T>
struct cl::sycl::detail::reserve_id< T >

A private description of a reservation station.

Definition at line 40 of file pipe.hpp.

Collaboration diagram for cl::sycl::detail::reserve_id< T >:

Public Member Functions

- reserve_id (typename boost::circular_buffer< T >::iterator start, std::size_t size)

  Track a reservation not committed yet.

Public Attributes

- boost::circular_buffer< T >::iterator start

  Start of the reservation in the pipe storage.
- std::size_t size

  Number of elements in the reservation.
- bool ready = false

8.1.2.12.1 Constructor & Destructor Documentation

8.1.2.12.1.1 reserve_id()

```
template<typename T >
cl::sycl::detail::reserve_id< T >::reserve_id (
            typename boost::circular_buffer< T >::iterator start,
            std::size_t size )  [inline]
```

Track a reservation not committed yet.

Parameters

in	*start*	point to the start of the reservation in the pipe storage
in	*size*	is the number of elements in the reservation

Definition at line 58 of file pipe.hpp.

```
00059                                : start { start }, size { size } {}
```

8.1.2.12.2 Member Data Documentation

8.1.2.12.2.1 ready

```
template<typename T >
bool cl::sycl::detail::reserve_id< T >::ready = false
```

Definition at line 49 of file pipe.hpp.

8.1.2.12.2.2 size

```
template<typename T >
std::size_t cl::sycl::detail::reserve_id< T >::size
```

Number of elements in the reservation.

Definition at line 45 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe< value_type >::empty(), cl::sycl::detail::pipe< value_type >::reserve_read(), cl::sycl::detail::pipe< value_type >::reserve_write(), and cl::sycl::detail::pipe< value_type >::size_with_lock().

8.1.2.12.2.3 start

```
template<typename T >
boost::circular_buffer<T>::iterator cl::sycl::detail::reserve_id< T >::start
```

Start of the reservation in the pipe storage.

Definition at line 42 of file pipe.hpp.

8.1.2.13 class cl::sycl::detail::pipe

template<typename T>
class cl::sycl::detail::pipe< T >

Implement a pipe object.

Use some mutable members so that the pipe object can be changed even when the accessors are captured in a lambda.

Definition at line 70 of file pipe.hpp.

Inheritance diagram for cl::sycl::detail::pipe< T >:

Collaboration diagram for cl::sycl::detail::pipe< T >:

Public Types

- using value_type = T
- using implementation_t = boost::circular_buffer< value_type >

  Implement the pipe with a circular buffer.
- using rid_iterator = typename decltype(w_rid_q)::iterator

Public Member Functions

- pipe (std::size_t capacity)

  Create a pipe as a circular buffer of the required capacity.
- std::size_t capacity () const

  Return the maximum number of elements that can fit in the pipe.
- std::size_t size_with_lock () const

  The size() method used outside needs to lock the datastructure.
- bool empty_with_lock () const

  The empty() method used outside needs to lock the datastructure.
- bool full_with_lock () const
- bool write (const T &value, bool blocking=false)

  Try to write a value to the pipe.
- bool read (T &value, bool blocking=false)

  Try to read a value from the pipe.
- std::size_t reserved_for_reading () const

  Compute the amount of elements blocked by read reservations, not yet committed.
- std::size_t reserved_for_writing () const

  Compute the amount of elements blocked by write reservations, not yet committed.
- bool reserve_read (std::size_t s, rid_iterator &rid, bool blocking=false)

  Reserve some part of the pipe for reading.
- bool reserve_write (std::size_t s, rid_iterator &rid, bool blocking=false)

  Reserve some part of the pipe for writing.
- void move_read_reservation_forward ()

  Process the read reservations that are ready to be released in the reservation queue.
- void move_write_reservation_forward ()

  Process the write reservations that are ready to be released in the reservation queue.

Public Attributes

- bool used_for_reading = false

  True when the pipe is currently used for reading.
- bool used_for_writing = false

  True when the pipe is currently used for writing.

Private Member Functions

- std::size_t size () const

  Get the current number of elements in the pipe that can be read.
- bool empty () const

  Test if the pipe is empty.
- bool full () const

  Test if the pipe is full.

Private Attributes

- boost::circular_buffer< value_type > cb

  The circular buffer to store the elements.
- std::mutex cb_mutex

  To protect the access to the circular buffer.
- std::deque< reserve_id< value_type > > w_rid_q

  The queue of pending write reservations.
- std::deque< reserve_id< value_type > > r_rid_q

  The queue of pending read reservations.
- std::size_t read_reserved_frozen

  Track the number of frozen elements related to read reservations.
- std::condition_variable read_done

  To signal that a read has been successful.
- std::condition_variable write_done

  To signal that a write has been successful.
- bool debug_mode = false

  To control the debug mode, disabled by default.

8.1.2.13.1 Member Typedef Documentation

8.1.2.13.1.1 implementation_t

```
template<typename T>
using cl::sycl::detail::pipe< T >::implementation_t = boost::circular_buffer<value_type>
```

Implement the pipe with a circular buffer.

Definition at line 77 of file pipe.hpp.

8.1.2.13.1.2 rid_iterator

```
template<typename T>
using cl::sycl::detail::pipe< T >::rid_iterator = typename decltype(w_rid_q)::iterator
```

Definition at line 95 of file pipe.hpp.

8.1.2.13.1.3 value_type

```
template<typename T>
using cl::sycl::detail::pipe< T >::value_type = T
```

Definition at line 74 of file pipe.hpp.

8.1.2.13.2 Constructor & Destructor Documentation

8.1.2.13.2.1 pipe()

```
template<typename T>
cl::sycl::detail::pipe< T >::pipe (
            std::size_t capacity )  [inline]
```

Create a pipe as a circular buffer of the required capacity.

Definition at line 126 of file pipe.hpp.

```
00126 : cb { capacity }, read_reserved_frozen { 0 } { }
```

8.1.2.13.3 Member Function Documentation

8.1.2.13.3.1 capacity()

```
template<typename T>
std::size_t cl::sycl::detail::pipe< T >::capacity ( ) const  [inline]
```

Return the maximum number of elements that can fit in the pipe.

Definition at line 131 of file pipe.hpp.

```
00131                                {
00132     // No lock required since it is fixed and set at construction time
00133     return cb.capacity();
00134   }
```

8.1.2.13.3.2 empty()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::empty ( ) const  [inline], [private]
```

Test if the pipe is empty.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the write side (for example on FPGA).

Definition at line 166 of file pipe.hpp.

```
00166                        {
00167     TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
00168                    << " size() = " << size());
00169     // It is empty when the size is zero, taking into account reservations
00170     return size() ==  0;
00171   }
```

8.1.2.13.3.3 empty_with_lock()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::empty_with_lock ( ) const  [inline]
```

The empty() method used outside needs to lock the datastructure.

Definition at line 197 of file pipe.hpp.

```
00197                                {
00198     std::lock_guard<std::mutex> lg { cb_mutex };
00199     return empty();
00200   }
```

8.1.2.13.3.4 full()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::full ( ) const  [inline], [private]
```

Test if the pipe is full.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the read side (for example on FPGA).

Definition at line 182 of file pipe.hpp.

```
00182                      {
00183     return cb.full();
00184   }
```

8.1.2.13.3.5 full_with_lock()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::full_with_lock ( ) const  [inline]
```

Definition at line 204 of file pipe.hpp.

```
00204                                {
00205     std::lock_guard<std::mutex> lg { cb_mutex };
00206     return full();
00207   }
```

8.1.2.13.3.6 move_read_reservation_forward()

```
template<typename T>
void cl::sycl::detail::pipe< T >::move_read_reservation_forward ( )  [inline]
```

Process the read reservations that are ready to be released in the reservation queue.

Definition at line 425 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::commit().

```
00425                                       {
00426     // Lock the pipe to avoid nuisance
00427     std::lock_guard<std::mutex> lg { cb_mutex };
00428
00429     for (;;) {
00430       if (r_rid_q.empty())
00431         // No pending reservation, so nothing to do
00432         break;
00433       if (!r_rid_q.front().ready)
00434         /* If the first reservation is not ready to be released, stop
00435            because it is blocking all the following in the queue
00436            anyway */
00437         break;
00438       // Remove the reservation to be released from the queue
00439       r_rid_q.pop_front();
00440       std::size_t n_to_pop;
00441       if (r_rid_q.empty())
00442         // If it was the last one, remove all the reservation
00443         n_to_pop = read_reserved_frozen;
00444       else
00445         // Else remove everything up to the next reservation
00446         n_to_pop =  r_rid_q.front().start - cb.begin();
00447       // No longer take into account these reserved slots
00448       read_reserved_frozen -= n_to_pop;
00449       // Release the elements from the FIFO
00450       while (n_to_pop--)
00451         cb.pop_front();
00452       // Notify the clients waiting for some room to write in the pipe
00453       read_done.notify_all();
00454       /* ...and process the next reservation to see if it is ready to
00455          be released too */
00456     }
00457   }
```

Here is the caller graph for this function:

8.1.2.13.3.7 move_write_reservation_forward()

```
template<typename T>
void cl::sycl::detail::pipe< T >::move_write_reservation_forward ( )  [inline]
```

Process the write reservations that are ready to be released in the reservation queue.

Definition at line 463 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::commit().

```
                                        {
    // Lock the pipe to avoid nuisance
    std::lock_guard<std::mutex> lg { cb_mutex };

    for (;;) {
      if (w_rid_q.empty())
        // No pending reservation, so nothing to do
        break;
      // Get the first reservation
      const auto &rid = w_rid_q.front();
      if (!rid.ready)
        /* If the reservation is not ready to be released, stop
           because it is blocking all the following in the queue
           anyway */
        break;
      // Remove the reservation to be released from the queue
      w_rid_q.pop_front();
      // Notify the clients waiting to read something from the pipe
      write_done.notify_all();
      /* ...and process the next reservation to see if it is ready to
         be released too */
    }
  }
```

Here is the caller graph for this function:

8.1.2.13.3.8 read()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::read (
            T & value,
            bool blocking = false )  [inline]
```

Try to read a value from the pipe.

Parameters

out	*value*	is the reference to where to store what is read
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true on success

If there is a pending reservation, read the next element to be read and update the number of reserved elements

Definition at line 258 of file pipe.hpp.

```
                                        {
    // Lock the pipe to avoid being disturbed
```

```
    std::unique_lock<std::mutex> ul { cb_mutex };
    TRISYCL_DUMP_T("Read pipe empty = " << empty());

    if (blocking)
      /* If in blocking mode, wait for the not empty condition, that
         may be changed when a write is done */
      write_done.wait(ul, [&] { return !empty(); });
    else if (empty())
      return false;

    TRISYCL_DUMP_T("Read pipe front = " << cb.front()
                   << " back = " << cb.back()
                   << " reserved_for_reading() = " << reserved_for_reading());
    if (read_reserved_frozen)
      /** If there is a pending reservation, read the next element to
          be read and update the number of reserved elements */
      value = cb.begin()[read_reserved_frozen++];
    else {
      /* There is no pending read reservation, so pop the read value
         from the pipe */
      value = cb.front();
      cb.pop_front();
    }

    TRISYCL_DUMP_T("Read pipe value = " << value);
    // Notify the clients waiting for some room to write in the pipe
    read_done.notify_all();
    return true;
  }
```

8.1.2.13.3.9 reserve_read()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::reserve_read (
            std::size_t s,
            rid_iterator & rid,
            bool blocking = false )  [inline]
```

Reserve some part of the pipe for reading.

Parameters

in	*s*	is the number of element to reserve
out	*rid*	is an iterator to a description of the reservation that has been done if successful
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true if the reservation was successful

Definition at line 335 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

```
                                                {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };

    TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    if (s == 0)
      // Empty reservation requested, so nothing to do
      return false;

```

```
    if (blocking)
      /* If in blocking mode, wait for enough elements to read in the
         pipe for the reservation. This condition can change when a
         write is done */
      write_done.wait(ul, [&] { return s <= size(); });
    else if (s > size())
      // Not enough elements to read in the pipe for the reservation
      return false;

    // Compute the location of the first element of the reservation
    auto first = cb.begin() + read_reserved_frozen;
    // Increment the number of frozen elements
    read_reserved_frozen += s;
    /* Add a description of the reservation at the end of the
       reservation queue */
    r_rid_q.emplace_back(first, s);
    // Return the iterator to the last reservation descriptor
    rid = r_rid_q.end() - 1;
    TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    return true;
  }
```

Here is the caller graph for this function:

8.1.2.13.3.10 reserve_write()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::reserve_write (
            std::size_t s,
            rid_iterator & rid,
            bool blocking = false )  [inline]
```

Reserve some part of the pipe for writing.

Parameters

in	*s*	is the number of element to reserve
out	*rid*	is an iterator to a description of the reservation that has been done if successful
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true if the reservation was successful

Definition at line 383 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

```
                                                {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };

    TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    if (s == 0)
      // Empty reservation requested, so nothing to do
      return false;

    if (blocking)
      /* If in blocking mode, wait for enough room in the pipe, that
         may be changed when a read is done. Do not use a difference
         here because it is only about unsigned values */
      read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
    else if (cb.size() + s > capacity())
      // Not enough room in the pipe for the reservation
      return false;

    /* If there is enough room in the pipe, just create default values
         in it to do the reservation */
    for (std::size_t i = 0; i != s; ++i)
      cb.push_back();
    /* Compute the location of the first element a posteriori since it
         may not exist a priori if cb was empty before */
    auto first = cb.end() - s;
    /* Add a description of the reservation at the end of the
       reservation queue */
    w_rid_q.emplace_back(first, s);
    // Return the iterator to the last reservation descriptor
    rid = w_rid_q.end() - 1;
    TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    return true;
  }
```

Here is the caller graph for this function:

8.1.2.13.3.11 reserved_for_reading()

```
template<typename T>
std::size_t cl::sycl::detail::pipe< T >::reserved_for_reading ( ) const  [inline]
```

Compute the amount of elements blocked by read reservations, not yet committed.

This includes some normal reads to pipes between/after un-committed reservations

This function assumes that the data structure is locked

Definition at line 299 of file pipe.hpp.

```
                                        {
    return read_reserved_frozen;
  }
```

8.1.2.13.3.12 reserved_for_writing()

```
template<typename T>
std::size_t cl::sycl::detail::pipe< T >::reserved_for_writing ( ) const  [inline]
```

Compute the amount of elements blocked by write reservations, not yet committed.

This includes some normal writes to pipes between/after un-committed reservations

This function assumes that the data structure is locked

Definition at line 312 of file pipe.hpp.

```
00312                                          {
00313     if (w_rid_q.empty())
00314       // No on-going reservation
00315       return 0;
00316     else
00317       /* The reserved size is from the first element of the first
00318          on-going reservation up to the end of the pipe content */
00319       return cb.end() - w_rid_q.front().start;
00320   }
```

8.1.2.13.3.13 size()

```
template<typename T>
std::size_t cl::sycl::detail::pipe< T >::size ( ) const  [inline], [private]
```

Get the current number of elements in the pipe that can be read.

This is obviously a volatile value which is constrained by the theory of restricted relativity.

Note that on some devices it may be costly to implement (for example on FPGA).

Definition at line 146 of file pipe.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::end(), and cl::sycl::detail::pipe_reservation< PipeAccessor >::size().

```
00146                           {
00147     TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
00148                    << " cb.end() = " << (void *)&*cb.end()
00149                    << " reserved_for_reading() = " << reserved_for_reading()
00150                    << " reserved_for_writing() = " << reserved_for_writing());
00151     /* The actual number of available elements depends from the
00152        elements blocked by some reservations.
00153        This prevents a consumer to read into reserved area. */
00154     return cb.size() - reserved_for_reading() -
      reserved_for_writing();
00155   }
```

Here is the caller graph for this function:

8.1.2.13.3.14 size_with_lock()

```
template<typename T>
std::size_t cl::sycl::detail::pipe< T >::size_with_lock ( ) const  [inline]
```

The size() method used outside needs to lock the datastructure.

Definition at line 190 of file pipe.hpp.

```
00190                                   {
00191     std::lock_guard<std::mutex> lg { cb_mutex };
00192     return size();
00193   }
```

8.1.2.13.3.15 write()

```
template<typename T>
bool cl::sycl::detail::pipe< T >::write (
            const T & value,
            bool blocking = false )  [inline]
```

Try to write a value to the pipe.

Parameters

in	*value*	is what we want to write
in	*blocking*	specify if the call wait for the operation to succeed

Returns

true on success

Todo provide a && version

Definition at line 221 of file pipe.hpp.

```
00221                                                   {
00222     // Lock the pipe to avoid being disturbed
00223     std::unique_lock<std::mutex> ul { cb_mutex };
00224     TRISYCL_DUMP_T("Write pipe full = " << full()
00225                    << " value = " << value);
00226
00227     if (blocking)
00228       /* If in blocking mode, wait for the not full condition, that
00229          may be changed when a read is done */
00230       read_done.wait(ul, [&] { return !full(); });
00231     else if (full())
00232       return false;
00233
00234     cb.push_back(value);
00235     TRISYCL_DUMP_T("Write pipe front = " << cb.front()
00236                    << " back = " << cb.back()
00237                    << " cb.begin() = " << (void *)&*cb.begin()
00238                    << " cb.size() = " << cb.size()
00239                    << " cb.end() = " << (void *)&*cb.end()
00240                    << " reserved_for_reading() = " << reserved_for_reading()
00241                    << " reserved_for_writing() = " << reserved_for_writing());
00242     // Notify the clients waiting to read something from the pipe
00243     write_done.notify_all();
00244     return true;
00245   }
```

8.1.2.13.4 Member Data Documentation

8.1.2.13.4.1 cb

```
template<typename T>
boost::circular_buffer<value_type> cl::sycl::detail::pipe< T >::cb  [private]
```

The circular buffer to store the elements.

Definition at line 82 of file pipe.hpp.

8.1.2.13.4.2 cb_mutex

```
template<typename T>
std::mutex cl::sycl::detail::pipe< T >::cb_mutex  [mutable], [private]
```

To protect the access to the circular buffer.

In case the object is capture in a lambda per copy, make it mutable.

Definition at line 88 of file pipe.hpp.

8.1.2.13.4.3 debug_mode

```
template<typename T>
bool cl::sycl::detail::pipe< T >::debug_mode = false  [private]
```

To control the debug mode, disabled by default.

Definition at line 115 of file pipe.hpp.

8.1.2.13.4.4 r_rid_q

```
template<typename T>
std::deque<reserve_id<value_type> > cl::sycl::detail::pipe< T >::r_rid_q  [private]
```

The queue of pending read reservations.

Definition at line 103 of file pipe.hpp.

8.1.2.13.4.5 read_done

```
template<typename T>
std::condition_variable cl::sycl::detail::pipe< T >::read_done [private]
```

To signal that a read has been successful.

Definition at line 109 of file pipe.hpp.

8.1.2.13.4.6 read_reserved_frozen

```
template<typename T>
std::size_t cl::sycl::detail::pipe< T >::read_reserved_frozen [private]
```

Track the number of frozen elements related to read reservations.

Definition at line 106 of file pipe.hpp.

8.1.2.13.4.7 used_for_reading

```
template<typename T>
bool cl::sycl::detail::pipe< T >::used_for_reading = false
```

True when the pipe is currently used for reading.

Definition at line 120 of file pipe.hpp.

8.1.2.13.4.8 used_for_writing

```
template<typename T>
bool cl::sycl::detail::pipe< T >::used_for_writing = false
```

True when the pipe is currently used for writing.

Definition at line 123 of file pipe.hpp.

8.1.2.13.4.9 w_rid_q

```
template<typename T>
std::deque<reserve_id<value_type> > cl::sycl::detail::pipe< T >::w_rid_q [private]
```

The queue of pending write reservations.

Definition at line 91 of file pipe.hpp.

8.1.2.13.4.10 write_done

```
template<typename T>
std::condition_variable cl::sycl::detail::pipe< T >::write_done [private]
```

To signal that a write has been successful.

Definition at line 112 of file pipe.hpp.

8.1.2.14 class cl::sycl::detail::pipe_accessor

template<typename T, access::mode AccessMode, access::target Target>
class cl::sycl::detail::pipe_accessor< T, AccessMode, Target >

The accessor abstracts the way pipe data are accessed inside a kernel.

Definition at line 44 of file pipe_accessor.hpp.

Inheritance diagram for cl::sycl::detail::pipe_accessor< T, AccessMode, Target >:

Collaboration diagram for cl::sycl::detail::pipe_accessor< T, AccessMode, Target >:

Public Types

- using value_type = T

  The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &

Public Member Functions

- pipe_accessor (const std::shared_ptr< detail::pipe< T >> &p, handler &command_group_handler)

  Construct a pipe accessor from an existing pipe.
- pipe_accessor ()=default
- std::size_t capacity () const

  Return the maximum number of elements that can fit in the pipe.
- std::size_t size () const

  Get the current number of elements in the pipe.
- bool empty () const

  Test if the pipe is empty.
- bool full () const

  Test if the pipe is full.
- operator bool () const

  In an explicit bool context, the accessor gives the success status of the last access.
- const pipe_accessor & write (const value_type &value) const

  Try to write a value to the pipe.
- const pipe_accessor & operator<< (const value_type &value) const

  Some syntactic sugar to use.
- const pipe_accessor & read (value_type &value) const

  Try to read a value from the pipe.
- value_type read () const

  Read a value from a blocking pipe.
- const pipe_accessor & operator>> (value_type &value) const

  Some syntactic sugar to use.
- detail::pipe_reservation< pipe_accessor > reserve (std::size_t size) const
- void set_debug (bool enable) const

  Set debug mode.
- auto & get_pipe_detail ()
- ~pipe_accessor ()

Static Public Attributes

- static constexpr auto rank = 1
- static constexpr auto mode = AccessMode
- static constexpr auto target = Target
- static constexpr bool blocking

Private Attributes

- std::shared_ptr< detail::pipe< T > > implementation

  The real pipe implementation behind the hood.
- bool ok = false

  Store the success status of last pipe operation.

8.1.2.14.1 Member Typedef Documentation

8.1.2.14.1.1 const_reference

```
template<typename T, access::mode AccessMode, access::target Target>
using cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::const_reference = const value↩
_type&
```

Definition at line 59 of file pipe_accessor.hpp.

8.1.2.14.1.2 reference

```
template<typename T, access::mode AccessMode, access::target Target>
using cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::reference = value_type&
```

Definition at line 58 of file pipe_accessor.hpp.

8.1.2.14.1.3 value_type

```
template<typename T, access::mode AccessMode, access::target Target>
using cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::value_type = T
```

The STL-like types.

Definition at line 57 of file pipe_accessor.hpp.

8.1.2.14.2 Constructor & Destructor Documentation

8.1.2.14.2.1 pipe_accessor() [1/2]

```
template<typename T, access::mode AccessMode, access::target Target>
cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor (
            const std::shared_ptr< detail::pipe< T >> & p,
            handler & command_group_handler )  [inline]
```

Construct a pipe accessor from an existing pipe.

Todo Use pipe_exception instead

Definition at line 83 of file pipe_accessor.hpp.

```
00084                                                  :
00085     implementation { p } {
00086     //    TRISYCL_DUMP_T("Create a kernel pipe accessor write = "
00087     //                   << is_write_access());
00088     // Verify that the pipe is not already used in the requested mode
00089     if (mode == access::mode::write)
00090       if (implementation->used_for_writing)
00091         /// \todo Use pipe_exception instead
00092         throw std::logic_error { "The pipe is already used for writing." };
00093       else
00094         implementation->used_for_writing = true;
00095     else
00096       if (implementation->used_for_reading)
00097         throw std::logic_error { "The pipe is already used for reading." };
00098       else
00099         implementation->used_for_reading = true;
00100   }
```

8.1.2.14.2.2 pipe_accessor() [2/2]

```
template<typename T, access::mode AccessMode, access::target Target>
cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::pipe_accessor ( ) [default]
```

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::pipe_accessor().

Here is the caller graph for this function:

8.1.2.14.2.3 ~pipe_accessor()

```
template<typename T, access::mode AccessMode, access::target Target>
cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::~pipe_accessor ( ) [inline]
```

Free the pipe for a future usage for the current mode

Definition at line 272 of file pipe_accessor.hpp.

```
00272                    {
00273     /// Free the pipe for a future usage for the current mode
00274     if (mode == access::mode::write)
00275       implementation->used_for_writing = false;
00276     else
00277       implementation->used_for_reading = false;
00278   }
```

8.1.2.14.3 Member Function Documentation

8.1.2.14.3.1 capacity()

```
template<typename T, access::mode AccessMode, access::target Target>
std::size_t cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::capacity ( ) const
[inline]
```

Return the maximum number of elements that can fit in the pipe.

Definition at line 107 of file pipe_accessor.hpp.

```
00107                              {
00108     return implementation->capacity();
00109   }
```

8.1.2.14.3.2 empty()

```
template<typename T, access::mode AccessMode, access::target Target>
bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::empty ( ) const  [inline]
```

Test if the pipe is empty.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the write side (for example on FPGA).

Definition at line 132 of file pipe_accessor.hpp.

```
00132                     {
00133     return implementation->empty_with_lock();
00134   }
```

8.1.2.14.3.3 full()

```
template<typename T, access::mode AccessMode, access::target Target>
bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::full ( ) const  [inline]
```

Test if the pipe is full.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement on the read side (for example on FPGA).

Definition at line 145 of file pipe_accessor.hpp.

```
00145                     {
00146     return implementation->full_with_lock();
00147   }
```

8.1.2.14.3.4 get_pipe_detail()

```
template<typename T, access::mode AccessMode, access::target Target>
auto& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::get_pipe_detail ( )  [inline]
```

Definition at line 267 of file pipe_accessor.hpp.

```
00267                           {
00268     return implementation;
00269   }
```

8.1.2.14.3.5 operator bool()

```
template<typename T, access::mode AccessMode, access::target Target>
cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::operator bool ( ) const  [inline],
[explicit]
```

In an explicit bool context, the accessor gives the success status of the last access.

It is not impacted by reservation success.

The explicitness is related to avoid

```
 some_pipe <<
some_value
```

to be interpreted as

```
 some_bool <<
some_value
```

when the type of

```
some_value
```

is not the same type as the pipe type.

Returns

true on success of the previous read or write operation

Definition at line 162 of file pipe_accessor.hpp.

```
00162                                   {
00163     return ok;
00164   }
```

8.1.2.14.3.6 operator<<()

```
template<typename T, access::mode AccessMode, access::target Target>
const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::operator<< (
            const value_type & value ) const  [inline]
```

Some syntactic sugar to use.

```
a << v
```

instead of

```
a.write(v)
```

Definition at line 192 of file pipe_accessor.hpp.

```
00192                                                                   {
00193     static_assert(mode == access::mode::write,
00194                   "'<<' operator on a pipe accessor is only possible"
00195                   " with write access mode");
00196     // Return a reference to *this so we can apply a sequence of >>
00197     return write(value);
00198   }
```

8.1.2.14.3.7 operator>>()

```
template<typename T, access::mode AccessMode, access::target Target>
const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::operator>> (
            value_type & value ) const  [inline]
```

Some syntactic sugar to use.

```
a >> v
```

instead of

```
a.read(v)
```

Definition at line 247 of file pipe_accessor.hpp.

```
00247                                                                   {
00248     static_assert(mode == access::mode::read,
00249                   "'>>' operator on a pipe accessor is only possible"
00250                   " with read access mode");
00251     // Return a reference to *this so we can apply a sequence of >>
00252     return read(value);
00253   }
```

8.1.2.14.3.8 read() [1/2]

```
template<typename T, access::mode AccessMode, access::target Target>
const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::read (
            value_type & value ) const  [inline]
```

Try to read a value from the pipe.

Parameters

out	*value*	is the reference to where to store what is read

Returns

`this`
so we can apply a sequence of read for example (but do not do this on a non blocking pipe...)

This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

Definition at line 213 of file pipe_accessor.hpp.

```
00213                                                           {
00214     static_assert(mode == access::mode::read,
00215                   "'.read(value_type &value)' method on a pipe accessor"
00216                   " is only possible with read access mode");
00217     ok = implementation->read(value, blocking);
00218     // Return a reference to *this so we can apply a sequence of read
00219     return *this;
00220   }
```

8.1.2.14.3.9 read() [2/2]

```
template<typename T, access::mode AccessMode, access::target Target>
value_type cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::read ( ) const  [inline]
```

Read a value from a blocking pipe.

Returns

the read value directly, since it cannot fail on blocking pipe

This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

Definition at line 232 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >↩ ::operator>>().

```
00232                                {
00233     static_assert(mode == access::mode::read,
00234                   "'.read()' method on a pipe accessor is only possible"
00235                   " with read access mode");
00236     static_assert(blocking,
00237                   "'.read()' method on a pipe accessor is only possible"
00238                   " with a blocking pipe");
00239     value_type value;
00240     implementation->read(value, blocking);
00241     return value;
00242   }
```

Here is the caller graph for this function:

8.1.2.14.3.10 reserve()

```
template<typename T, access::mode AccessMode, access::target Target>
detail::pipe_reservation<pipe_accessor> cl::sycl::detail::pipe_accessor< T, AccessMode, Target
>::reserve (
            std::size_t size ) const  [inline]
```

Definition at line 256 of file pipe_accessor.hpp.

```
00256                                                                    {
00257     return { *implementation, size };
00258   }
```

8.1.2.14.3.11 set_debug()

```
template<typename T, access::mode AccessMode, access::target Target>
void cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::set_debug (
            bool enable ) const  [inline]
```

Set debug mode.

Definition at line 262 of file pipe_accessor.hpp.

```
00262                                    {
00263     implementation->debug_mode = enable;
00264   }
```

8.1.2.14.3.12 size()

```
template<typename T, access::mode AccessMode, access::target Target>
std::size_t cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::size ( ) const  [inline]
```

Get the current number of elements in the pipe.

This is obviously a volatile value which is constrained by restricted relativity.

Note that on some devices it may be costly to implement (for example on FPGA).

Definition at line 119 of file pipe_accessor.hpp.

```
00119                           {
00120     return implementation->size_with_lock();
00121   }
```

8.1.2.14.3.13 write()

```
template<typename T, access::mode AccessMode, access::target Target>
const pipe_accessor& cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::write (
            const value_type & value ) const  [inline]
```

Try to write a value to the pipe.

Parameters

in	*value*	is what we want to write

Returns

this so we can apply a sequence of write for example (but do not do this on a non blocking pipe...)

Todo provide a && version

This function is const so it can work when the accessor is passed by copy in the [=] kernel lambda, which is not mutable by default

Definition at line 180 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >↩ ::operator<<().

```
00180                                                                   {
00181     static_assert(mode == access::mode::write,
00182                   "'.write(const value_type &value)' method on a pipe accessor"
00183                   " is only possible with write access mode");
00184     ok = implementation->write(value, blocking);
00185     // Return a reference to *this so we can apply a sequence of write
00186     return *this;
00187   }
```

Here is the caller graph for this function:

8.1.2.14.4 Member Data Documentation

8.1.2.14.4.1 blocking

```
template<typename T, access::mode AccessMode, access::target Target>
constexpr bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::blocking  [static]
```

Initial value:

```
=
    (target == cl::sycl::access::target::blocking_pipe)
```

Definition at line 53 of file pipe_accessor.hpp.

8.1.2.14.4.2 implementation

```
template<typename T, access::mode AccessMode, access::target Target>
std::shared_ptr<detail::pipe<T> > cl::sycl::detail::pipe_accessor< T, AccessMode, Target >↩
::implementation  [private]
```

The real pipe implementation behind the hood.

Definition at line 64 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::get↩ _pipe_detail(), and cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >↩ ::reserve().

8.1.2.14.4.3 mode

```
template<typename T, access::mode AccessMode, access::target Target>
constexpr auto cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::mode = AccessMode
[static]
```

Definition at line 50 of file pipe_accessor.hpp.

8.1.2.14.4.4 ok

```
template<typename T, access::mode AccessMode, access::target Target>
bool cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::ok = false  [mutable], [private]
```

Store the success status of last pipe operation.

It is not impacted by reservation success.

It does exist even if the pipe accessor is not evaluated in a boolean context for, but a use-def analysis can optimise it out in that case and not use some storage

Use a mutable state here so that it can work with a [=] lambda capture without having to declare the whole lambda as mutable

Definition at line 77 of file pipe_accessor.hpp.

Referenced by cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::operator bool().

8.1.2.14.4.5 rank

```
template<typename T, access::mode AccessMode, access::target Target>
constexpr auto cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::rank = 1  [static]
```

Definition at line 49 of file pipe_accessor.hpp.

8.1.2.14.4.6 target

```
template<typename T, access::mode AccessMode, access::target Target>
constexpr auto cl::sycl::detail::pipe_accessor< T, AccessMode, Target >::target = Target
[static]
```

Definition at line 51 of file pipe_accessor.hpp.

8.1.2.15 class cl::sycl::pipe

template<typename T>
class cl::sycl::pipe< T >

A SYCL pipe.

Implement a FIFO-style object that can be used through accessors to send some objects T from the input to the output

Definition at line 31 of file accessor.hpp.

Inheritance diagram for cl::sycl::pipe< T >:

Collaboration diagram for cl::sycl::pipe< T >:

Public Types

- using value_type = T

  The STL-like types.

Public Member Functions

- pipe (std::size_t capacity)

  Construct a pipe able to store up to capacity T objects.
- template<access::mode Mode, access::target Target = access::target::pipe>
  accessor< value_type, 1, Mode, Target > get_access (handler &command_group_handler)

  Get an accessor to the pipe with the required mode.
- std::size_t capacity () const

  Return the maximum number of elements that can fit in the pipe.

Private Types

- using implementation_t = typename pipe::shared_ptr_implementation

Private Attributes

- friend implementation_t

Additional Inherited Members

8.1.2.15.1 Member Typedef Documentation

8.1.2.15.1.1 implementation_t

```
template<typename T>
using cl::sycl::pipe< T >::implementation_t = typename pipe::shared_ptr_implementation  [private]
```

Definition at line 40 of file pipe.hpp.

8.1.2.15.1.2 value_type

```
template<typename T>
using cl::sycl::pipe< T >::value_type = T
```

The STL-like types.

Definition at line 53 of file pipe.hpp.

8.1.2.15.2 Constructor & Destructor Documentation

8.1.2.15.2.1 pipe()

```
template<typename T>
cl::sycl::pipe< T >::pipe (
            std::size_t capacity )  [inline]
```

Construct a pipe able to store up to capacity T objects.

Definition at line 57 of file pipe.hpp.

References cl::sycl::access::pipe.

```
00058     : implementation_t { new detail::pipe<T> { capacity } } { }
```

8.1.2.15.3 Member Function Documentation

8.1.2.15.3.1 capacity()

```
template<typename T>
std::size_t cl::sycl::pipe< T >::capacity ( ) const  [inline]
```

Return the maximum number of elements that can fit in the pipe.

Definition at line 83 of file pipe.hpp.

References cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation.

```
00083                                {
00084     return implementation->capacity();
00085     }
```

8.1.2.15.3.2 get_access()

```
template<typename T>
template<access::mode Mode, access::target Target = access::target::pipe>
accessor<value_type, 1, Mode, Target> cl::sycl::pipe< T >::get_access (
            handler & command_group_handler )  [inline]
```

Get an accessor to the pipe with the required mode.

Parameters

	Mode	is the requested access mode
	Target	is the type of pipe access required
in	*command_group_handler*	is the command group handler in which the kernel is to be executed

Definition at line 73 of file pipe.hpp.

References cl::sycl::access::blocking_pipe, cl::sycl::detail::shared_ptr_implementation< pipe< T >, detail::pipe< T > >::implementation, and cl::sycl::access::pipe.

```
00073                                                {
00074     static_assert(Target == access::target::pipe
00075                   || Target == access::target::blocking_pipe,
00076                   "get_access(handler) with pipes can only deal with "
00077                   "access::pipe or access::blocking_pipe");
00078     return { implementation, command_group_handler };
00079   }
```

8.1.2.15.4 Member Data Documentation

8.1.2.15.4.1 implementation_t

```
template<typename T>
friend cl::sycl::pipe< T >::implementation_t  [private]
```

Definition at line 43 of file pipe.hpp.

8.1.2.16 class cl::sycl::detail::pipe_reservation

template<typename PipeAccessor>
class cl::sycl::detail::pipe_reservation< PipeAccessor >

The implementation of the pipe reservation station.

Definition at line 33 of file pipe_reservation.hpp.

Inheritance diagram for cl::sycl::detail::pipe_reservation< PipeAccessor >:

Collaboration diagram for cl::sycl::detail::pipe_reservation< PipeAccessor >:

Public Types

- using iterator = typename detail::pipe< value_type >::implementation_t::iterator
- using const_iterator = typename detail::pipe< value_type >::implementation_t::const_iterator

Public Member Functions

- void assume_validity ()

  Test that the reservation is in a usable state.
- pipe_reservation (detail::pipe< value_type > &p, std::size_t s)

  Create a pipe reservation station that reserves the pipe itself.
- pipe_reservation (const pipe_reservation &)=delete

  No copy constructor with some spurious commit in the destructor of the original object.
- pipe_reservation (pipe_reservation &&orig)

  Only a move constructor is required to move it into the shared_ptr.
- pipe_reservation ()=default

  Keep the default constructors too.
- operator bool ()

  Test if the reservation succeeded and thus if the reservation can be committed.
- iterator begin ()

  Start of the reservation area.
- iterator end ()

  Past the end of the reservation area.
- std::size_t size ()

  Get the number of elements in the reservation station.
- reference operator[ ] (std::size_t index)

  Access to an element of the reservation.
- void commit ()

  Commit the reservation station.
- ~pipe_reservation ()

  An implicit commit is made in the destructor.

Public Attributes

- bool ok = false

  True if the reservation was successful and still uncommitted.
- detail::pipe< value_type >::rid_iterator rid

  Point into the reservation buffer. Only valid if ok is true.
- detail::pipe< value_type > & p

  Keep a reference on the pipe to access to the data and methods.

Static Public Attributes

- static constexpr access::mode mode = accessor_type::mode
- static constexpr access::target target = accessor_type::target

Private Types

- using accessor_type = PipeAccessor
- using value_type = typename accessor_type::value_type
- using reference = typename accessor_type::reference

Static Private Attributes

- static constexpr bool blocking

8.1.2.16.1 Member Typedef Documentation

8.1.2.16.1.1 accessor_type

```
template<typename PipeAccessor>
using cl::sycl::detail::pipe_reservation< PipeAccessor >::accessor_type = PipeAccessor  [private]
```

Definition at line 35 of file pipe_reservation.hpp.

8.1.2.16.1.2 const_iterator

```
template<typename PipeAccessor>
using cl::sycl::detail::pipe_reservation< PipeAccessor >::const_iterator = typename detail↩
::pipe<value_type>::implementation_t::const_iterator
```

Definition at line 46 of file pipe_reservation.hpp.

8.1.2.16.1.3 iterator

```
template<typename PipeAccessor>
using cl::sycl::detail::pipe_reservation< PipeAccessor >::iterator = typename detail::pipe<value↩
_type>::implementation_t::iterator
```

Definition at line 44 of file pipe_reservation.hpp.

8.1.2.16.1.4 reference

```
template<typename PipeAccessor>
using cl::sycl::detail::pipe_reservation< PipeAccessor >::reference = typename accessor_type↩
::reference  [private]
```

Definition at line 39 of file pipe_reservation.hpp.

8.1.2.16.1.5 value_type

```
template<typename PipeAccessor>
using cl::sycl::detail::pipe_reservation< PipeAccessor >::value_type = typename accessor_↩
type::value_type  [private]
```

Definition at line 38 of file pipe_reservation.hpp.

8.1.2.16.2 Constructor & Destructor Documentation

8.1.2.16.2.1 pipe_reservation() [1/4]

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (
            detail::pipe< value_type > & p,
            std::size_t s )  [inline]
```

Create a pipe reservation station that reserves the pipe itself.

Definition at line 78 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation(), cl::sycl::access::read, cl::sycl↩ ::detail::pipe< T >::reserve_read(), cl::sycl::detail::pipe< T >::reserve_write(), and cl::sycl::access::write.

```
00078                                                          : p { p } {
00079     static_assert(mode == access::mode::write
00080                   || mode == access::mode::read,
00081                   "A pipe can only be accesed in read or write mode,"
00082                   " exclusively");
00083
00084     /* Since this test is constexpr and dependent of a template
00085        parameter, it should be equivalent to a specialization of the
00086        method but in a clearer way */
00087     if (mode == access::mode::write)
00088       ok = p.reserve_write(s, rid, blocking);
00089     else
00090       ok = p.reserve_read(s, rid, blocking);
00091   }
```

Here is the call graph for this function:

8.1.2.16.2.2 pipe_reservation() [2/4]

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (
            const pipe_reservation< PipeAccessor > &  )  [delete]
```

No copy constructor with some spurious commit in the destructor of the original object.

8.1.2.16.2.3 pipe_reservation() [3/4]

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation (
            pipe_reservation< PipeAccessor > && orig )  [inline]
```

Only a move constructor is required to move it into the shared_ptr.

Definition at line 101 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

```
00101                                                 :
00102     ok {orig.ok },
00103     rid {orig.rid },
00104     p { orig.p } {
00105       /* Even when an object is moved, the destructor of the old
00106          object is eventually called, so leave the old object in a
00107          destructable state but without any commit capability */
00108       orig.ok = false;
00109     }
```

Here is the call graph for this function:

8.1.2.16.2.4 pipe_reservation() [4/4]

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation ( )  [default]
```

Keep the default constructors too.

Otherwise there is no move semantics and the copy is made by creating a new reservation and destructing the old one with a spurious commit in the meantime...

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::pipe_reservation().

Here is the caller graph for this function:

8.1.2.16.2.5 ~pipe_reservation()

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::~pipe_reservation ( ) [inline]
```

An implicit commit is made in the destructor.

Definition at line 185 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::commit().

```
00185                        {
00186     commit();
00187   }
```

Here is the call graph for this function:

8.1.2.16.3 Member Function Documentation

8.1.2.16.3.1 assume_validity()

```
template<typename PipeAccessor>
void cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity ( ) [inline]
```

Test that the reservation is in a usable state.

Todo Throw exception instead

Definition at line 71 of file pipe_reservation.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::begin(), cl::sycl::detail::pipe_reservation< PipeAccessor >::end(), cl::sycl::detail::pipe_reservation< PipeAccessor >::operator[]() , and cl::sycl::detail↩ ::pipe_reservation< PipeAccessor >::size().

```
00071                          {
00072     assert(ok);
00073   }
```

Here is the caller graph for this function:

8.1.2.16.3.2 begin()

```
template<typename PipeAccessor>
iterator cl::sycl::detail::pipe_reservation< PipeAccessor >::begin ( )  [inline]
```

Start of the reservation area.

Definition at line 134 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity().

```
00134                    {
00135      assume_validity();
00136      return rid->start;
00137    }
```

Here is the call graph for this function:

8.1.2.16.3.3 commit()

```
template<typename PipeAccessor>
void cl::sycl::detail::pipe_reservation< PipeAccessor >::commit ( ) [inline]
```

Commit the reservation station.

Todo Add to the specification that for simplicity a reservation can be commited several times but only the first one is taken into account

Definition at line 170 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe< T >::move_read_reservation_forward(), cl::sycl::detail::pipe< T >::move_write↩_reservation_forward(), TRISYCL_DUMP_T, and cl::sycl::access::write.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::~pipe_reservation().

```
00170                 {
00171     if (ok) {
00172       // If the reservation is in a committable state, commit
00173       TRISYCL_DUMP_T("Commit");
00174       rid->ready = true;
00175       if (mode == access::mode::write)
00176         p.move_write_reservation_forward();
00177       else
00178         p.move_read_reservation_forward();
00179       ok = false;
00180     }
00181   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.1.2.16.3.4 end()

```
template<typename PipeAccessor>
iterator cl::sycl::detail::pipe_reservation< PipeAccessor >::end ( ) [inline]
```

Past the end of the reservation area.

Definition at line 141 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity(), and cl::sycl::detail::pipe< T >↩ ::size().

```
00141                 {
00142     assume_validity();
00143     return rid->start + rid->size;
00144   }
```

Here is the call graph for this function:

8.1.2.16.3.5 operator bool()

```
template<typename PipeAccessor>
cl::sycl::detail::pipe_reservation< PipeAccessor >::operator bool ( ) [inline]
```

Test if the reservation succeeded and thus if the reservation can be committed.

Note that it is up to the user to ensure that all the reservation elements have been initialized correctly in the case of a write for example

Definition at line 128 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::ok.

```
00128                 {
00129     return ok;
00130   }
```

8.1.2.16.3.6 operator[]()

```
template<typename PipeAccessor>
reference cl::sycl::detail::pipe_reservation< PipeAccessor >::operator[] (
            std::size_t index )  [inline]
```

Access to an element of the reservation.

Definition at line 155 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity(), and TRISYCL_DUMP_T.

```
00155                                        {
00156     assume_validity();
00157     TRISYCL_DUMP_T("[] index = " << index
00158                    << " Reservation write address = " << &(rid->start[index]));
00159
00160     return rid->start[index];
00161   }
```

Here is the call graph for this function:

8.1.2.16.3.7 size()

```
template<typename PipeAccessor>
std::size_t cl::sycl::detail::pipe_reservation< PipeAccessor >::size ( )  [inline]
```

Get the number of elements in the reservation station.

Definition at line 148 of file pipe_reservation.hpp.

References cl::sycl::detail::pipe_reservation< PipeAccessor >::assume_validity(), and cl::sycl::detail::pipe< T >↩ ::size().

```
00148                    {
00149     assume_validity();
00150     return rid->size;
00151   }
```

Here is the call graph for this function:

8.1.2.16.4 Member Data Documentation

8.1.2.16.4.1 blocking

```
template<typename PipeAccessor>
constexpr bool cl::sycl::detail::pipe_reservation< PipeAccessor >::blocking [static], [private]
```

Initial value:

```
=
    (accessor_type::target ==
      cl::sycl::access::target::blocking_pipe)
```

Definition at line 36 of file pipe_reservation.hpp.

8.1.2.16.4.2 mode

```
template<typename PipeAccessor>
constexpr access::mode cl::sycl::detail::pipe_reservation< PipeAccessor >::mode = accessor_↩
type::mode [static]
```

Definition at line 49 of file pipe_reservation.hpp.

8.1.2.16.4.3 ok

```
template<typename PipeAccessor>
bool cl::sycl::detail::pipe_reservation< PipeAccessor >::ok = false
```

True if the reservation was successful and still uncommitted.

B default a pipe_reservation is not reserved and cannot be committed

Definition at line 55 of file pipe_reservation.hpp.

Referenced by cl::sycl::detail::pipe_reservation< PipeAccessor >::operator bool().

8.1.2.16.4.4 p

```
template<typename PipeAccessor>
detail::pipe<value_type>& cl::sycl::detail::pipe_reservation< PipeAccessor >::p
```

Keep a reference on the pipe to access to the data and methods.

Note that with inlining and CSE it should not use more register when compiler optimization is in use.

Definition at line 64 of file pipe_reservation.hpp.

8.1.2.16.4.5 rid

```
template<typename PipeAccessor>
detail::pipe<value_type>::rid_iterator cl::sycl::detail::pipe_reservation< PipeAccessor >↩
::rid
```

Point into the reservation buffer. Only valid if ok is true.

Definition at line 58 of file pipe_reservation.hpp.

8.1.2.16.4.6 target

```
template<typename PipeAccessor>
constexpr access::target cl::sycl::detail::pipe_reservation< PipeAccessor >::target = accessor↩
_type::target  [static]
```

Definition at line 50 of file pipe_reservation.hpp.

8.1.2.17 struct cl::sycl::pipe_reservation

template<typename PipeAccessor>
struct cl::sycl::pipe_reservation< PipeAccessor >

The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example.

Definition at line 30 of file pipe_reservation.hpp.

Collaboration diagram for cl::sycl::pipe_reservation< PipeAccessor >:

Public Types

- using accessor_type = PipeAccessor
- using accessor_detail = typename accessor_type::accessor_detail
- using value_type = typename accessor_type::value_type

  The STL-like types.
- using reference = value_type &
- using const_reference = const value_type &
- using pointer = value_type ∗
- using const_pointer = const value_type ∗
- using size_type = std::size_t
- using difference_type = ptrdiff_t
- using iterator = typename detail::pipe_reservation< accessor_detail >::iterator
- using const_iterator = typename detail::pipe_reservation< accessor_detail >::const_iterator
- using reverse_iterator = std::reverse_iterator< iterator >
- using const_reverse_iterator = std::reverse_iterator< const_iterator >

Public Member Functions

- pipe_reservation ()=default

  Use default constructors so that we can create a new buffer copy from another one, with either a l-value or a r-value (for std::move() for example).
- pipe_reservation (accessor_type &accessor, std::size_t s)

  Create a pipe_reservation for an accessor and a number of elements.
- pipe_reservation (detail::pipe_reservation< accessor_detail > &&pr)

  Create a pipe_reservation from the implementation detail.
- operator bool () const

  Test if the pipe_reservation has been correctly allocated.
- std::size_t size () const

  Get the number of reserved element(s)
- reference operator[ ] (std::size_t index) const

  Access to a given element of the reservation.
- void commit () const

  Force a commit operation.
- iterator begin () const

  Get an iterator on the first element of the reservation station.
- iterator end () const

  Get an iterator past the end of the reservation station.
- const_iterator cbegin () const

  Build a constant iterator on the first element of the reservation station.
- const_iterator cend () const

  Build a constant iterator past the end of the reservation station.
- reverse_iterator rbegin () const

  Get a reverse iterator on the last element of the reservation station.
- reverse_iterator rend () const

  Get a reverse iterator on the first element past the end of the reservation station.
- const_reverse_iterator crbegin () const

  Get a constant reverse iterator on the last element of the reservation station.
- const_reverse_iterator crend () const

  Get a constant reverse iterator on the first element past the end of the reservation station.

Public Attributes

- std::shared_ptr< detail::pipe_reservation< accessor_detail > > implementation

  Point to the underlying implementation that can be shared in the SYCL model with a handler semantics.

Static Public Attributes

- static constexpr bool blocking

8.1.2.17.1 Member Typedef Documentation

8.1.2.17.1.1 accessor_detail

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::accessor_detail = typename accessor_type↩
::accessor_detail
```

Definition at line 34 of file pipe_reservation.hpp.

8.1.2.17.1.2 accessor_type

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::accessor_type = PipeAccessor
```

Definition at line 31 of file pipe_reservation.hpp.

8.1.2.17.1.3 const_iterator

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::const_iterator = typename detail::pipe_↩
reservation<accessor_detail>::const_iterator
```

Definition at line 46 of file pipe_reservation.hpp.

8.1.2.17.1.4 const_pointer

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::const_pointer = const value_type*
```

Definition at line 40 of file pipe_reservation.hpp.

8.1.2.17.1.5 const_reference

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::const_reference = const value_type&
```

Definition at line 38 of file pipe_reservation.hpp.

8.1.2.17.1.6 const_reverse_iterator

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::const_reverse_iterator = std::reverse_↩
iterator<const_iterator>
```

Definition at line 48 of file pipe_reservation.hpp.

8.1.2.17.1.7 difference_type

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::difference_type = ptrdiff_t
```

Definition at line 42 of file pipe_reservation.hpp.

8.1.2.17.1.8 iterator

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::iterator = typename detail::pipe_reservation<accessor↩
_detail>::iterator
```

Definition at line 44 of file pipe_reservation.hpp.

8.1.2.17.1.9 pointer

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::pointer = value_type*
```

Definition at line 39 of file pipe_reservation.hpp.

8.1.2.17.1.10 reference

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::reference = value_type&
```

Definition at line 37 of file pipe_reservation.hpp.

8.1.2.17.1.11 reverse_iterator

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::reverse_iterator = std::reverse_iterator<iterator>
```

Definition at line 47 of file pipe_reservation.hpp.

8.1.2.17.1.12 size_type

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::size_type = std::size_t
```

Definition at line 41 of file pipe_reservation.hpp.

8.1.2.17.1.13 value_type

```
template<typename PipeAccessor >
using cl::sycl::pipe_reservation< PipeAccessor >::value_type = typename accessor_type::value↩
_type
```

The STL-like types.

Definition at line 36 of file pipe_reservation.hpp.

8.1.2.17.2 Constructor & Destructor Documentation

8.1.2.17.2.1 pipe_reservation() [1/3]

```
template<typename PipeAccessor >
cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation ( ) [default]
```

Use default constructors so that we can create a new buffer copy from another one, with either a l-value or a r-value (for std::move() for example).

Since we just copy the shared_ptr<> above, this is where/how the sharing magic is happening with reference counting in this case.

8.1.2.17.2.2 pipe_reservation() [2/3]

```
template<typename PipeAccessor >
cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (
            accessor_type & accessor,
            std::size_t s )  [inline]
```

Create a pipe_reservation for an accessor and a number of elements.

Definition at line 66 of file pipe_reservation.hpp.

References cl::sycl::get_pipe_detail().

```
00067     : implementation {
00068     new detail::pipe_reservation<accessor_detail> {
00069       get_pipe_detail(accessor), s }
00070   } {}
```

Here is the call graph for this function:

8.1.2.17.2.3 pipe_reservation() [3/3]

```
template<typename PipeAccessor >
cl::sycl::pipe_reservation< PipeAccessor >::pipe_reservation (
            detail::pipe_reservation< accessor_detail > && pr )  [inline]
```

Create a pipe_reservation from the implementation detail.

This is an internal constructor to allow reserve() on the implementation to lift a full-fledged object through accessor↩ ::reserve().

Todo Make it private and add required friends

Definition at line 81 of file pipe_reservation.hpp.

```
00082     : implementation {
00083     new detail::pipe_reservation<accessor_detail> { std::move(pr) } }
00084   {}
```

8.1.2.17.3 Member Function Documentation

8.1.2.17.3.1 begin()

```
template<typename PipeAccessor >
iterator cl::sycl::pipe_reservation< PipeAccessor >::begin ( ) const  [inline]
```

Get an iterator on the first element of the reservation station.

Definition at line 119 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::rend().

```
00119                                {
00120     return implementation->begin();
00121   }
```

Here is the caller graph for this function:

8.1.2.17.3.2 cbegin()

```
template<typename PipeAccessor >
const_iterator cl::sycl::pipe_reservation< PipeAccessor >::cbegin ( ) const  [inline]
```

Build a constant iterator on the first element of the reservation station.

Definition at line 131 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::crend().

```
00131                                      {
00132     return implementation->begin();
00133   }
```

Here is the caller graph for this function:

8.1.2.17.3.3 cend()

```
template<typename PipeAccessor >
const_iterator cl::sycl::pipe_reservation< PipeAccessor >::cend ( ) const  [inline]
```

Build a constant iterator past the end of the reservation station.

Definition at line 137 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::crbegin().

```
00137                                {
00138     return implementation->end();
00139   }
```

Here is the caller graph for this function:

8.1.2.17.3.4 commit()

```
template<typename PipeAccessor >
void cl::sycl::pipe_reservation< PipeAccessor >::commit ( ) const  [inline]
```

Force a commit operation.

Normally the commit is implicitly done in the destructor, but sometime it is useful to do it earlier.

Definition at line 113 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
00113                        {
00114     return implementation->commit();
00115   }
```

8.1.2.17.3.5 crbegin()

```
template<typename PipeAccessor >
const_reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::crbegin ( ) const  [inline]
```

Get a constant reverse iterator on the last element of the reservation station.

Definition at line 157 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::cend().

```
00157                                        {
00158     return std::make_reverse_iterator(cend());
00159   }
```

Here is the call graph for this function:

8.1.2.17.3.6 crend()

```
template<typename PipeAccessor >
const_reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::crend ( ) const  [inline]
```

Get a constant reverse iterator on the first element past the end of the reservation station.

Definition at line 164 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::cbegin().

```
00164                                        {
00165     return std::make_reverse_iterator(cbegin());
00166   }
```

Here is the call graph for this function:

8.1.2.17.3.7 end()

```
template<typename PipeAccessor >
iterator cl::sycl::pipe_reservation< PipeAccessor >::end ( ) const  [inline]
```

Get an iterator past the end of the reservation station.

Definition at line 125 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::rbegin().

```
00125                        {
00126     return implementation->end();
00127   }
```

Here is the caller graph for this function:

8.1.2.17.3.8 operator bool()

```
template<typename PipeAccessor >
cl::sycl::pipe_reservation< PipeAccessor >::operator bool ( ) const  [inline]
```

Test if the pipe_reservation has been correctly allocated.

Returns

true if the pipe_reservation can be used and committed

Definition at line 91 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
00091                        {
00092     return *implementation;
00093   }
```

8.1.2.17.3.9 operator[]()

```
template<typename PipeAccessor >
reference cl::sycl::pipe_reservation< PipeAccessor >::operator[] (
            std::size_t index ) const  [inline]
```

Access to a given element of the reservation.

Definition at line 103 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
00103                                                {
00104     return (*implementation)[index];
00105   }
```

8.1.2.17.3.10 rbegin()

```
template<typename PipeAccessor >
reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::rbegin ( ) const  [inline]
```

Get a reverse iterator on the last element of the reservation station.

Definition at line 143 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::end().

```
00143                                    {
00144     return std::make_reverse_iterator(end());
00145   }
```

Here is the call graph for this function:

8.1.2.17.3.11 rend()

```
template<typename PipeAccessor >
reverse_iterator cl::sycl::pipe_reservation< PipeAccessor >::rend ( ) const  [inline]
```

Get a reverse iterator on the first element past the end of the reservation station.

Definition at line 150 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::begin().

```
00150                                {
00151     return std::make_reverse_iterator(begin());
00152   }
```

Here is the call graph for this function:

8.1.2.17.3.12 size()

```
template<typename PipeAccessor >
std::size_t cl::sycl::pipe_reservation< PipeAccessor >::size ( ) const  [inline]
```

Get the number of reserved element(s)

Definition at line 97 of file pipe_reservation.hpp.

References cl::sycl::pipe_reservation< PipeAccessor >::implementation.

```
00097                          {
00098     return implementation->size();
00099   }
```

8.1.2.17.4 Member Data Documentation

8.1.2.17.4.1 blocking

```
template<typename PipeAccessor >
constexpr bool cl::sycl::pipe_reservation< PipeAccessor >::blocking [static]
```

Initial value:

```
=
    (accessor_type::target ==
      cl::sycl::access::target::blocking_pipe)
```

Definition at line 32 of file pipe_reservation.hpp.

8.1.2.17.4.2 implementation

```
template<typename PipeAccessor >
std::shared_ptr<detail::pipe_reservation<accessor_detail> > cl::sycl::pipe_reservation<
PipeAccessor >::implementation
```

Point to the underlying implementation that can be shared in the SYCL model with a handler semantics.

Definition at line 53 of file pipe_reservation.hpp.

Referenced by cl::sycl::pipe_reservation< PipeAccessor >::begin(), cl::sycl::pipe_reservation< PipeAccessor >::cbegin(), cl::sycl::pipe_reservation< PipeAccessor >::cend(), cl::sycl::pipe_reservation< PipeAccessor >↩::commit(), cl::sycl::pipe_reservation< PipeAccessor >::end(), cl::sycl::pipe_reservation< PipeAccessor >↩::operator bool(), cl::sycl::pipe_reservation< PipeAccessor >::operator[ ](), and cl::sycl::pipe_reservation< Pipe↩Accessor >::size().

8.1.2.18 class cl::sycl::static_pipe

template<typename T, std::size_t Capacity>
class cl::sycl::static_pipe< T, Capacity >

A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe.

Implement a FIFO-style object that can be used through accessors to send some objects T from the input to the output.

Compared to a normal pipe, a static_pipe takes a constexpr size and is expected to be declared in a compile-unit static context so the compiler can generate everything at compile time.

This is useful to generate a fixed and optimized hardware implementation on FPGA for example, where the interconnection graph can be also inferred at compile time.

It is not directly mapped to the OpenCL program-scoped pipe because in SYCL there is not this concept of separated program. But the SYCL device compiler is expected to generate some OpenCL program(s) with program-scoped pipes when a SYCL static-scoped pipe is used. These details are implementation defined.

Definition at line 50 of file static_pipe.hpp.

Inheritance diagram for cl::sycl::static_pipe< T, Capacity >:

Collaboration diagram for cl::sycl::static_pipe< T, Capacity >:

Public Types

- using value_type = T

  The STL-like types.

Public Member Functions

- static_pipe ()

  Construct a static-scoped pipe able to store up to Capacity T objects.
- template<access::mode Mode, access::target Target = access::target::pipe>
  accessor< value_type, 1, Mode, Target > get_access (handler &command_group_handler)

  Get an accessor to the pipe with the required mode.
- std::size_t constexpr capacity () const

  Return the maximum number of elements that can fit in the pipe.

Private Types

- using implementation_t = typename static_pipe::shared_ptr_implementation

Private Attributes

- friend implementation_t

Additional Inherited Members

8.1.2.18.1 Member Typedef Documentation

8.1.2.18.1.1 implementation_t

```
template<typename T , std::size_t Capacity>
using cl::sycl::static_pipe< T, Capacity >::implementation_t = typename static_pipe::shared_↩
ptr_implementation  [private]
```

Definition at line 58 of file static_pipe.hpp.

8.1.2.18.1.2 value_type

```
template<typename T , std::size_t Capacity>
using cl::sycl::static_pipe< T, Capacity >::value_type = T
```

The STL-like types.

Definition at line 69 of file static_pipe.hpp.

8.1.2.18.2 Constructor & Destructor Documentation

8.1.2.18.2.1 static_pipe()

```
template<typename T , std::size_t Capacity>
cl::sycl::static_pipe< T, Capacity >::static_pipe ( )  [inline]
```

Construct a static-scoped pipe able to store up to Capacity T objects.

Definition at line 73 of file static_pipe.hpp.

References cl::sycl::access::pipe.

```
00074     : implementation_t { new detail::pipe<T> { Capacity } } { }
```

8.1.2.18.3 Member Function Documentation

8.1.2.18.3.1 capacity()

```
template<typename T , std::size_t Capacity>
std::size_t constexpr cl::sycl::static_pipe< T, Capacity >::capacity ( ) const  [inline]
```

Return the maximum number of elements that can fit in the pipe.

This is a constexpr since the capacity is in the type.

Definition at line 102 of file static_pipe.hpp.

```
00102                                       {
00103     return Capacity;
00104     }
```

8.1.2.18.3.2 get_access()

```
template<typename T , std::size_t Capacity>
template<access::mode Mode, access::target Target = access::target::pipe>
accessor<value_type, 1, Mode, Target> cl::sycl::static_pipe< T, Capacity >::get_access (
            handler & command_group_handler )  [inline]
```

Get an accessor to the pipe with the required mode.

Parameters

	Mode	is the requested access mode
	Target	is the type of pipe access required
in	*command_group_handler*	is the command group handler in which the kernel is to be executed

Definition at line 89 of file static_pipe.hpp.

References cl::sycl::access::blocking_pipe, cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >::implementation, and cl::sycl::access::pipe.

```
00089                                                {
00090     static_assert(Target == access::target::pipe
00091                   || Target == access::target::blocking_pipe,
00092                   "get_access(handler) with pipes can only deal with "
00093                   "access::pipe or access::blocking_pipe");
00094     return { implementation, command_group_handler };
00095   }
```

8.1.2.18.4 Member Data Documentation

8.1.2.18.4.1 implementation_t

```
template<typename T , std::size_t Capacity>
friend cl::sycl::static_pipe< T, Capacity >::implementation_t  [private]
```

Definition at line 64 of file static_pipe.hpp.

8.1.3 Typedef Documentation

8.1.3.1 buffer_allocator

```
template<typename T >
using cl::sycl::buffer_allocator = typedef std::allocator<T>
```

`#include <include/CL/sycl/allocator.hpp>`

The allocator objects give the programmer some control on how the memory is allocated inside SYCL.

The default buffer allocator used by the runtime, when no allocator is defined by the user.

The allocator used for the `buffer` inside SYCL

Just use the default allocator for now.

Reuse the C++ default allocator.

Definition at line 30 of file allocator.hpp.

8.1.3.2 image_allocator

```
template<typename T >
using cl::sycl::image_allocator = typedef std::allocator<T>
```

#include <include/CL/sycl/allocator.hpp>

The allocator used for the `image` inside SYCL.

Just use the default allocator for now.

Definition at line 38 of file allocator.hpp.

8.1.3.3 map_allocator

```
template<typename T >
using cl::sycl::map_allocator = typedef std::allocator<T>
```

#include <include/CL/sycl/allocator.hpp>

The allocator used to map the memory at the same place.

Just use the default allocator for now.

Todo : implement and clarify the specification. It looks like it is not really an allocator according the current spec

Definition at line 49 of file allocator.hpp.

8.1.4 Function Documentation

8.1.4.1 add_buffer_to_task()

```
static std::shared_ptr<detail::task> cl::sycl::detail::add_buffer_to_task (
            handler * command_group_handler,
            std::shared_ptr< detail::buffer_base > b,
            bool is_write_mode )  [inline], [static]
```

#include <include/CL/sycl/buffer/detail/buffer_base.hpp>

Referenced by cl::sycl::detail::buffer_base::add_to_task().

Here is the caller graph for this function:

8.1.4.2 buffer_add_to_task()

```
template<typename BufferDetail >
static std::shared_ptr<detail::task> cl::sycl::detail::buffer_add_to_task (
            BufferDetail buf,
            handler * command_group_handler,
            bool is_write_mode )  [static]
```

#include <include/CL/sycl/buffer/detail/buffer.hpp>

Proxy function to avoid some circular type recursion.

Returns

a shared_ptr<task>

Todo To remove with some refactoring

Definition at line 436 of file buffer.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor().

```
00438                                       {
00439     return buf->add_to_task(command_group_handler, is_write_mode);
00440   }
```

Here is the caller graph for this function:

8.1.4.3 get_pipe_detail()

```
template<typename Accessor >
static auto& cl::sycl::get_pipe_detail (
            Accessor & a )  [inline], [static]
```

#include <include/CL/sycl/accessor.hpp>

Top-level function to break circular dependencies on the the types to get the pipe implementation.

Definition at line 481 of file accessor.hpp.

Referenced by cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >::get_pipe_detail(), cl::sycl↩ ::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >::get_pipe_detail(), and cl::sycl::pipe_↩ reservation< PipeAccessor >::pipe_reservation().

```
00481                                                   {
00482   return a.get_pipe_detail();
00483   }
```

Here is the caller graph for this function:

8.1.4.4 waiter()

```
template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_↩
const_t<T>>>
auto cl::sycl::detail::waiter (
            detail::buffer< T, Dimensions > * b )  [inline]
```

`#include <include/CL/sycl/buffer/detail/buffer_waiter.hpp>`

Helper function to create a new buffer_waiter.

Definition at line 79 of file buffer_waiter.hpp.

Referenced by cl::sycl::buffer< T, Dimensions, Allocator >::buffer().

```
00079                                                   {
00080   return new buffer_waiter<T, Dimensions, Allocator> { b };
00081 }
```

Here is the caller graph for this function:

8.2 Dealing with OpenCL address spaces

Collaboration diagram for Dealing with OpenCL address spaces:

Namespaces

- cl::sycl

Classes

- struct cl::sycl::detail::ocl_type< T, AS >

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct cl::sycl::detail::ocl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct cl::sycl::detail::ocl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct cl::sycl::detail::ocl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct cl::sycl::detail::ocl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_base< T, AS >

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct cl::sycl::detail::address_space_variable< T, AS >

  Implementation of a variable with an OpenCL address space. More...

Typedefs

- template<typename T , address_space AS>
  using cl::sycl::detail::addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.
- template<typename T >
  using cl::sycl::constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using cl::sycl::constant_ptr = constant< T * >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using cl::sycl::generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be in the OpenCL 2 generic address space.
- template<typename T >
  using cl::sycl::global = detail::addr_space< T, global_address_space >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using cl::sycl::global_ptr = global< T * >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using cl::sycl::local = detail::addr_space< T, local_address_space >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using cl::sycl::local_ptr = local< T * >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using cl::sycl::priv = detail::addr_space< T, private_address_space >

  Declare a variable to be in the OpenCL private address space.
- template<typename T >
  using cl::sycl::private_ptr = priv< T * >

  Declare a variable to be in the OpenCL private address space.
- template<typename Pointer , address_space AS>
  using cl::sycl::multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.

Enumerations

- enum cl::sycl::address_space {
  cl::sycl::constant_address_space, cl::sycl::generic_address_space, cl::sycl::global_address_space, cl::sycl::local_address_space,
  cl::sycl::private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename T , address_space AS>
  multi_ptr< T, AS > cl::sycl::make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.

8.2.1 Detailed Description

8.2.2 Class Documentation

8.2.2.1 struct cl::sycl::detail::ocl_type

template<typename T, address_space AS>
struct cl::sycl::detail::ocl_type< T, AS >

Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device.

In the general case, do not add any OpenCL address space qualifier

Definition at line 65 of file address_space.hpp.

Public Types

- using type = T

8.2.2.1.1 Member Typedef Documentation

8.2.2.1.1.1 type

```
template<typename T, address_space AS>
using cl::sycl::detail::ocl_type< T, AS >::type = T
```

Definition at line 66 of file address_space.hpp.

8.2.2.2 struct cl::sycl::detail::ocl_type< T, constant_address_space >

template<typename T>
struct cl::sycl::detail::ocl_type< T, constant_address_space >

Add an attribute for __constant address space.

Definition at line 71 of file address_space.hpp.

Public Types

- using type = TRISYCL_CONSTANT_AS T

8.2.2.2.1 Member Typedef Documentation

8.2.2.2.1.1 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, constant_address_space >::type = TRISYCL_CONSTANT_AS T
```

Definition at line 72 of file address_space.hpp.

8.2.2.3 struct cl::sycl::detail::ocl_type< T, global_address_space >

template<typename T>
struct cl::sycl::detail::ocl_type< T, global_address_space >

Add an attribute for __global address space.

Definition at line 85 of file address_space.hpp.

Public Types

- using type = TRISYCL_GLOBAL_AS T

8.2.2.3.1 Member Typedef Documentation

8.2.2.3.1.1 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, global_address_space >::type = TRISYCL_GLOBAL_AS T
```

Definition at line 86 of file address_space.hpp.

8.2.2.4 struct cl::sycl::detail::ocl_type< T, local_address_space >

template<typename T>
struct cl::sycl::detail::ocl_type< T, local_address_space >

Add an attribute for __local address space.

Definition at line 91 of file address_space.hpp.

Public Types

- using type = TRISYCL_LOCAL_AS T

8.2.2.4.1 Member Typedef Documentation

8.2.2.4.1.1 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, local_address_space >::type = TRISYCL_LOCAL_AS T
```

Definition at line 92 of file address_space.hpp.

8.2.2.5 struct cl::sycl::detail::ocl_type< T, private_address_space >

template<typename T>
struct cl::sycl::detail::ocl_type< T, private_address_space >

Add an attribute for __private address space.

Definition at line 97 of file address_space.hpp.

Public Types

- using type = TRISYCL_PRIVATE_AS T

8.2.2.5.1 Member Typedef Documentation

8.2.2.5.1.1 type

```
template<typename T >
using cl::sycl::detail::ocl_type< T, private_address_space >::type = TRISYCL_PRIVATE_AS T
```

Definition at line 98 of file address_space.hpp.

8.2.2.6 struct cl::sycl::detail::address_space_array

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_array< T, AS >

Implementation of an array variable with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

Definition at line 105 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_array< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_array< T, AS >:

Public Types

- using super = address_space_variable< T, AS >

Keep track of the base class as a short-cut.

Public Member Functions

- address_space_array (const T &array)

  Allow to create an address space array from an array.
- address_space_array (std::initializer_list< std::remove_extent_t< T >> list)

  Allow to create an address space array from an initializer list.

Additional Inherited Members

8.2.2.6.1 Member Typedef Documentation

8.2.2.6.1.1 super

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_array< T, AS >::super = address_space_variable<T, AS>
```

Keep track of the base class as a short-cut.

Definition at line 321 of file address_space.hpp.

8.2.2.6.2 Constructor & Destructor Documentation

8.2.2.6.2.1 address_space_array() [1/2]

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_array< T, AS >::address_space_array (
            const T & array )  [inline]
```

Allow to create an address space array from an array.

Definition at line 329 of file address_space.hpp.

```
00329                                    {
00330     std::copy(std::begin(array), std::end(array), std::begin(super::variable));
00331   };
```

8.2.2.6.2.2 address_space_array() [2/2]

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_array< T, AS >::address_space_array (
            std::initializer_list< std::remove_extent_t< T >> list )  [inline]
```

Allow to create an address space array from an initializer list.

Todo Extend to more than 1 dimension

Definition at line 338 of file address_space.hpp.

```
00338                                                                    {
00339     std::copy(std::begin(list), std::end(list), std::begin(super::variable));
00340   };
```

8.2.2.7 struct cl::sycl::detail::address_space_fundamental

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_fundamental< T, AS >

Implementation of a fundamental type with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

Todo Verify/improve to deal with const/volatile?

Definition at line 108 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_fundamental< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_fundamental< T, AS >:

Public Types

- using super = address_space_variable< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_fundamental ()=default

  Also request for the default constructors that have been disabled by the declaration of another constructor.

- template<typename SomeType , cl::sycl::address_space SomeAS>
  address_space_fundamental (address_space_fundamental< SomeType, SomeAS > &v)

  Allow for example assignment of a global<float> to a priv<double> for example.

Additional Inherited Members

8.2.2.7.1 Member Typedef Documentation

8.2.2.7.1.1 super

```
template<typename T, address_space AS>
using cl::sycl::detail::address_space_fundamental< T, AS >::super = address_space_variable<T,
AS>
```

Keep track of the base class as a short-cut.

Definition at line 229 of file address_space.hpp.

8.2.2.7.2 Constructor & Destructor Documentation

8.2.2.7.2.1 address_space_fundamental() [1/2]

```
template<typename T, address_space AS>
cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental ( ) [default]
```

Also request for the default constructors that have been disabled by the declaration of another constructor.

This ensures for example that we can write

```
generic<float *> q;
```

without initialization.

8.2.2.7.2.2 address_space_fundamental() [2/2]

```
template<typename T, address_space AS>
template<typename SomeType , cl::sycl::address_space SomeAS>
cl::sycl::detail::address_space_fundamental< T, AS >::address_space_fundamental (
            address_space_fundamental< SomeType, SomeAS > & v ) [inline]
```

Allow for example assignment of a global<float> to a priv<double> for example.

Since it needs 2 implicit conversions, it does not work with the conversion operators already define, so add 1 more explicit conversion here so that the remaining implicit conversion can be found by the compiler.

Strangely

```
template <typename SomeType, address_space SomeAS>
address_space_base(addr_space<SomeType, SomeAS>& v)
: variable(SomeType(v)) { }
```

cannot be used here because SomeType cannot be inferred. So use address_space_base<> instead

Need to think further about it...

Definition at line 267 of file address_space.hpp.

```
00268   {
00269     /* Strangely I cannot have it working in the initializer instead, for
00270        some cases */
00271     super::variable = SomeType(v);
00272   }
```

8.2.2.8 struct cl::sycl::detail::address_space_object

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_object< T, AS >

Implementation of an object type with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

The class implementation is just inheriting of T so that all methods and non-member operators on T work also on address_space_object<T>

Todo Verify/improve to deal with const/volatile?

Todo what about T having some final methods?

Definition at line 111 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_object< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_object< T, AS >:

Public Types

- using opencl_type = typename ocl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.

Public Member Functions

- address_space_object (T &&v)

  Allow to create an address space version of an object or to convert one.
- operator opencl_type & ()

  Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Additional Inherited Members

8.2.2.8.1 Member Typedef Documentation

8.2.2.8.1.1 opencl_type

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_object< T, AS >::opencl_type = typename ocl_type<T,
AS>::type
```

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 366 of file address_space.hpp.

8.2.2.8.2 Constructor & Destructor Documentation

8.2.2.8.2.1 address_space_object()

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_object< T, AS >::address_space_object (
            T && v )  [inline]
```

Allow to create an address space version of an object or to convert one.

Definition at line 377 of file address_space.hpp.

```
00377 : opencl_type(v) { }
```

8.2.2.8.3 Member Function Documentation

8.2.2.8.3.1 operator opencl_type &()

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_object< T, AS >::operator opencl_type & ( ) [inline]
```

Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Use opencl_type so that if we take the address of it, the address space is kept.

Definition at line 385 of file address_space.hpp.

```
00385 { return *this; }
```

8.2.2.9 struct cl::sycl::detail::address_space_ptr

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_ptr< T, AS >

Implementation for an OpenCL address space pointer.

Parameters

T	is the pointer type

Note that if *T* is not a pointer type, it is an error.

All the address space pointers inherit from it, which makes trivial the implementation of `cl::sycl::multi_ptr<T, AS>`

Definition at line 114 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_ptr< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_ptr< T, AS >:

Public Types

- using super = address_space_fundamental< T, AS >

  Keep track of the base class as a short-cut.
- using pointer_t = typename super::address_space_fundamental::type
- using reference_t = typename std::remove_pointer_t< pointer_t > &

Public Member Functions

- address_space_ptr (address_space_fundamental< typename std::pointer_traits< T >::element_type, AS > *p)

  Allow initialization of a pointer type from the address of an element with the same type and address space.

Additional Inherited Members

8.2.2.9.1 Member Typedef Documentation

8.2.2.9.1.1 pointer_t

```
template<typename T, address_space AS>
using cl::sycl::detail::address_space_ptr< T, AS >::pointer_t = typename super::address_↩
space_fundamental::type
```

Definition at line 298 of file address_space.hpp.

8.2.2.9.1.2 reference_t

```
template<typename T, address_space AS>
using cl::sycl::detail::address_space_ptr< T, AS >::reference_t = typename std::remove_↩
pointer_t<pointer_t>&
```

Definition at line 299 of file address_space.hpp.

8.2.2.9.1.3 super

```
template<typename T, address_space AS>
using cl::sycl::detail::address_space_ptr< T, AS >::super = address_space_fundamental<T, AS>
```

Keep track of the base class as a short-cut.

Definition at line 293 of file address_space.hpp.

8.2.2.9.2 Constructor & Destructor Documentation

8.2.2.9.2.1 address_space_ptr()

```
template<typename T, address_space AS>
cl::sycl::detail::address_space_ptr< T, AS >::address_space_ptr (
            address_space_fundamental< typename std::pointer_traits< T >::element_type, AS >
* p )  [inline]
```

Allow initialization of a pointer type from the address of an element with the same type and address space.

Definition at line 304 of file address_space.hpp.

References p.

```
00305     : address_space_fundamental<T, AS> { p->get_address() } {}
```

8.2.2.10 struct cl::sycl::detail::address_space_base

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_base< T, AS >

Implementation of the base infrastructure to wrap something in an OpenCL address space.

Parameters

T	is the type of the basic stuff to be created
AS	is the address space to place the object into

Todo Verify/improve to deal with const/volatile?

Definition at line 145 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_base< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_base< T, AS >:

Public Types

- using type = T

  Store the base type of the object.
- using opencl_type = typename ocl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.

Static Public Attributes

- static auto constexpr address_space = AS

  Set the address_space identifier that can be queried to know the pointer type.

8.2.2.10.1 Member Typedef Documentation

8.2.2.10.1.1 opencl_type

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_base< T, AS >::opencl_type = typename ocl_type<T, AS>↩
::type
```

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 156 of file address_space.hpp.

8.2.2.10.1.2 type

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_base< T, AS >::type = T
```

Store the base type of the object.

Todo Add to the specification

Definition at line 150 of file address_space.hpp.

8.2.2.10.2 Member Data Documentation

8.2.2.10.2.1 address_space

```
template<typename T , address_space AS>
auto constexpr cl::sycl::detail::address_space_base< T, AS >::address_space = AS  [static]
```

Set the address_space identifier that can be queried to know the pointer type.

Definition at line 160 of file address_space.hpp.

8.2.2.11 struct cl::sycl::detail::address_space_variable

template<typename T, address_space AS>
struct cl::sycl::detail::address_space_variable< T, AS >

Implementation of a variable with an OpenCL address space.

Parameters

T	is the type of the basic object to be created
AS	is the address space to place the object into

Definition at line 172 of file address_space.hpp.

Inheritance diagram for cl::sycl::detail::address_space_variable< T, AS >:

Collaboration diagram for cl::sycl::detail::address_space_variable< T, AS >:

Public Types

- using opencl_type = typename ocl_type< T, AS >::type

  Store the base type of the object with OpenCL address space modifier.
- using super = address_space_base< T, AS >

  Keep track of the base class as a short-cut.

Public Member Functions

- address_space_variable (const T &v)

  Allow to create an address space version of an object or to convert one to be used by the classes inheriting by this one because it is not possible to directly initialize a base class member in C++.
- address_space_variable ()=default

  Put back the default constructors canceled by the previous definition.
- operator opencl_type & ()

  Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.
- opencl_type * get_address ()

  Return the address of the value to implement pointers.

Protected Attributes

- opencl_type variable

Additional Inherited Members

8.2.2.11.1 Member Typedef Documentation

8.2.2.11.1.1 opencl_type

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_variable< T, AS >::opencl_type = typename ocl_type<T,
AS>::type
```

Store the base type of the object with OpenCL address space modifier.

Todo Add to the specification

Definition at line 177 of file address_space.hpp.

8.2.2.11.1.2 super

```
template<typename T , address_space AS>
using cl::sycl::detail::address_space_variable< T, AS >::super = address_space_base<T, AS>
```

Keep track of the base class as a short-cut.

Definition at line 180 of file address_space.hpp.

8.2.2.11.2 Constructor & Destructor Documentation

8.2.2.11.2.1 address_space_variable() [1/2]

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_variable< T, AS >::address_space_variable (
            const T & v )  [inline]
```

Allow to create an address space version of an object or to convert one to be used by the classes inheriting by this one because it is not possible to directly initialize a base class member in C++.

Definition at line 196 of file address_space.hpp.

```
00196 : variable(v) { }
```

8.2.2.11.2.2 address_space_variable() [2/2]

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_variable< T, AS >::address_space_variable ( )  [default]
```

Put back the default constructors canceled by the previous definition.

8.2.2.11.3 Member Function Documentation

8.2.2.11.3.1 get_address()

```
template<typename T , address_space AS>
opencl_type* cl::sycl::detail::address_space_variable< T, AS >::get_address ( )  [inline]
```

Return the address of the value to implement pointers.

Definition at line 213 of file address_space.hpp.

```
00213 { return &variable; }
```

8.2.2.11.3.2 operator opencl_type &()

```
template<typename T , address_space AS>
cl::sycl::detail::address_space_variable< T, AS >::operator opencl_type & ( )  [inline]
```

Conversion operator to allow a address_space_object<T> to be used as a T so that all the methods of a T and the built-in operators for T can be used on a address_space_object<T> too.

Use opencl_type so that if we take the address of it, the address space is kept.

Definition at line 210 of file address_space.hpp.

```
00210 { return variable; }
```

8.2.2.11.4 Member Data Documentation

8.2.2.11.4.1 variable

```
template<typename T , address_space AS>
opencl_type cl::sycl::detail::address_space_variable< T, AS >::variable  [protected]
```

Definition at line 189 of file address_space.hpp.

8.2.3 Typedef Documentation

8.2.3.1 addr_space

```
template<typename T , address_space AS>
using cl::sycl::detail::addr_space = typedef typename std::conditional<std::is_pointer<T>↩
::value, address_space_ptr<T, AS>, typename std::conditional<std::is_class<T>::value, address↩
_space_object<T, AS>, typename std::conditional<std::is_array<T>::value, address_space_↩
array<T, AS>, address_space_fundamental<T, AS> >::type>::type>::type
```

`#include <include/CL/sycl/address_space/detail/address_space.hpp>`

Dispatch the address space implementation according to the requested type.

Parameters

T	is the type of the object to be created
AS	is the address space to place the object into or to point to in the case of a pointer type

Definition at line 132 of file address_space.hpp.

8.2.3.2 constant

```
template<typename T >
using cl::sycl::constant = typedef detail::addr_space<T, constant_address_space>
```

`#include <include/CL/sycl/address_space.hpp>`

Declare a variable to be in the OpenCL constant address space.

Parameters

T	is the type of the object

Definition at line 55 of file address_space.hpp.

8.2.3.3 constant_ptr

```
template<typename T >
using cl::sycl::constant_ptr = typedef constant<T*>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL constant address space.

Parameters

T	is the type of the object

Definition at line 63 of file address_space.hpp.

8.2.3.4 generic

```
template<typename T >
using cl::sycl::generic = typedef detail::addr_space<T, generic_address_space>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL 2 generic address space.

Parameters

T	is the type of the object

Definition at line 71 of file address_space.hpp.

8.2.3.5 global

```
template<typename T >
using cl::sycl::global = typedef detail::addr_space<T, global_address_space>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL global address space.

Parameters

T	is the type of the object

Definition at line 79 of file address_space.hpp.

8.2.3.6 global_ptr

```
template<typename T >
using cl::sycl::global_ptr = typedef global<T*>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL global address space.

Parameters

T	is the type of the object

Definition at line 88 of file address_space.hpp.

8.2.3.7 local

```
template<typename T >
using cl::sycl::local = typedef detail::addr_space<T, local_address_space>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL local address space.

Parameters

T	is the type of the object

Definition at line 96 of file address_space.hpp.

8.2.3.8 local_ptr

```
template<typename T >
using cl::sycl::local_ptr = typedef local<T*>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL local address space.

Parameters

T	is the type of the object

Definition at line 104 of file address_space.hpp.

8.2.3.9 multi_ptr

```
template<typename Pointer , address_space AS>
using cl::sycl::multi_ptr = typedef detail::address_space_ptr<Pointer, AS>
```

```
#include <include/CL/sycl/address_space.hpp>
```

A pointer that can be statically associated to any address-space.

Parameters

Pointer	is the pointer type
AS	is the address space to point to

Note that if *Pointer* is not a pointer type, it is an error.

Definition at line 132 of file address_space.hpp.

8.2.3.10 priv

```
template<typename T >
using cl::sycl::priv = typedef detail::addr_space<T, private_address_space>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL private address space.

Parameters

T	is the type of the object

Definition at line 112 of file address_space.hpp.

8.2.3.11 private_ptr

```
template<typename T >
using cl::sycl::private_ptr = typedef priv<T*>
```

```
#include <include/CL/sycl/address_space.hpp>
```

Declare a variable to be in the OpenCL private address space.

Parameters

T	is the type of the object

Definition at line 120 of file address_space.hpp.

8.2.4 Enumeration Type Documentation

8.2.4.1 address_space

```
enum cl::sycl::address_space
```

```
#include <include/CL/sycl/address_space.hpp>
```

Enumerate the different OpenCL 2 address spaces.

Enumerator

constant_address_space	
generic_address_space	
global_address_space	
local_address_space	
private_address_space	

Definition at line 27 of file address_space.hpp.

```
00027                    {
00028   constant_address_space,
00029   generic_address_space,
00030   global_address_space,
00031   local_address_space,
00032   private_address_space,
00033 };
```

8.2.5 Function Documentation

8.2.5.1 make_multi()

```
template<typename T , address_space AS>
multi_ptr<T, AS> cl::sycl::make_multi (
            multi_ptr< T, AS > pointer )
```

```
#include <include/CL/sycl/address_space.hpp>
```

Construct a cl::sycl::multi_ptr<> with the right type.

Parameters

pointer	is the address with its address space to point to

Todo Implement the case with a plain pointer

Definition at line 142 of file address_space.hpp.

```
00142                                                   {
00143   return pointer;
00144 }
```

8.3 Platforms, contexts, devices and queues

SYCL host context.

Collaboration diagram for Platforms, contexts, devices and queues:

Namespaces

- cl::sycl::detail
- cl::sycl::info

Classes

- class cl::sycl::detail::context
- class cl::sycl::detail::host_context
- class cl::sycl::context

  SYCL context. More...
- class cl::sycl::detail::device

  An abstract class representing various models of SYCL devices. More...
- class cl::sycl::device

  SYCL device. More...
- class cl::sycl::device_type_selector

  A device selector by device_type. More...
- class cl::sycl::device_typename_selector< DeviceType >

  Select a device by template device_type parameter. More...
- class cl::sycl::device_selector

  The SYCL heuristics to select a device. More...
- class cl::sycl::handler

  Command group handler class. More...
- class cl::sycl::detail::kernel

  Abstract SYCL kernel. More...
- class cl::sycl::kernel

  SYCL kernel. More...
- class cl::sycl::detail::host_platform

  SYCL host platform. More...
- class cl::sycl::detail::opencl_platform

  SYCL OpenCL platform. More...
- class cl::sycl::detail::platform

  An abstract class representing various models of SYCL platforms. More...
- class cl::sycl::platform

  Abstract the OpenCL platform. More...
- class cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...

Typedefs

- using cl::sycl::default_selector = device_typename_selector< info::device_type::automatic >

  Devices selected by heuristics of the system.
- using cl::sycl::gpu_selector = device_typename_selector< info::device_type::gpu >

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.
- using cl::sycl::cpu_selector = device_typename_selector< info::device_type::cpu >

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.
- using cl::sycl::host_selector = device_typename_selector< info::device_type::host >

  Selects the SYCL host CPU device that does not require an OpenCL runtime.
- using cl::sycl::info::device_fp_config = unsigned int
- using cl::sycl::info::device_exec_capabilities = unsigned int
- using cl::sycl::info::device_queue_properties = unsigned int

Enumerations

- enum cl::sycl::info::context : int { cl::sycl::info::context::reference_count, cl::sycl::info::context::platform, cl::sycl::info::context::devices }

  Context information descriptors.
- enum cl::sycl::info::device_type : unsigned int { cl::sycl::info::device_type::cpu, cl::sycl::info::device_type::gpu, cl::sycl::info::device_type::accelerator, cl::sycl::info::device_type::custom, cl::sycl::info::device_type::automatic, cl::sycl::info::device_type::host, cl::sycl::info::device_type::opencl, cl::sycl::info::device_type::all }

  Type of devices.
- enum cl::sycl::info::device : int { cl::sycl::info::device::device_type, cl::sycl::info::device::vendor_id, cl::sycl::info::device::max_compute_units, cl::sycl::info::device::max_work_item_dimensions, cl::sycl::info::device::max_work_item_sizes, cl::sycl::info::device::max_work_group_size, cl::sycl::info::device::preferred_vector_width_char, cl::sycl::info::device::preferred_vector_width_short, cl::sycl::info::device::preferred_vector_width_int, cl::sycl::info::device::preferred_vector_width_long_long, cl::sycl::info::device::preferred_vector_width_float, cl::sycl::info::device::preferred_vector_width_double, cl::sycl::info::device::preferred_vector_width_half, cl::sycl::info::device::native_vector_witdth_char, cl::sycl::info::device::native_vector_witdth_short, cl::sycl::info::device::native_vector_witdth_int, cl::sycl::info::device::native_vector_witdth_long_long, cl::sycl::info::device::native_vector_witdth_float, cl::sycl::info::device::native_vector_witdth_double, cl::sycl::info::device::native_vector_witdth_half, cl::sycl::info::device::max_clock_frequency, cl::sycl::info::device::address_bits, cl::sycl::info::device::max_mem_alloc_size, cl::sycl::info::device::image_support, cl::sycl::info::device::max_read_image_args, cl::sycl::info::device::max_write_image_args, cl::sycl::info::device::image2d_max_height, cl::sycl::info::device::image2d_max_width, cl::sycl::info::device::image3d_max_height, cl::sycl::info::device::image3d_max_widht, cl::sycl::info::device::image3d_mas_depth, cl::sycl::info::device::image_max_buffer_size, cl::sycl::info::device::image_max_array_size, cl::sycl::info::device::max_samplers, cl::sycl::info::device::max_parameter_size, cl::sycl::info::device::mem_base_addr_align, cl::sycl::info::device::single_fp_config, cl::sycl::info::device::double_fp_config, cl::sycl::info::device::global_mem_cache_type, cl::sycl::info::device::global_mem_cache_line_size, cl::sycl::info::device::global_mem_cache_size, cl::sycl::info::device::global_mem_size, cl::sycl::info::device::max_constant_buffer_size, cl::sycl::info::device::max_constant_args, cl::sycl::info::device::local_mem_type, cl::sycl::info::device::local_mem_size, cl::sycl::info::device::error_correction_support, cl::sycl::info::device::host_unified_memory, cl::sycl::info::device::profiling_timer_resolution, cl::sycl::info::device::endian_little, cl::sycl::info::device::is_available, cl::sycl::info::device::is_compiler_available, cl::sycl::info::device::is_linker_available, cl::sycl::info::device::execution_capabilities, cl::sycl::info::device::queue_properties, cl::sycl::info::device::built_in_kernels,

cl::sycl::info::device::platform, cl::sycl::info::device::name, cl::sycl::info::device::vendor, cl::sycl::info::device::driver_version, cl::sycl::info::device::profile, cl::sycl::info::device::device_version, cl::sycl::info::device::opencl_version, cl::sycl::info::device::extensions, cl::sycl::info::device::printf_buffer_size, cl::sycl::info::device::preferred_interop_user_sync, cl::sycl::info::device::parent_device, cl::sycl::info::device::partition_max_sub_devices, cl::sycl::info::device::partition_properties, cl::sycl::info::device::partition_affinity_domain, cl::sycl::info::device::partition_type, cl::sycl::info::device::reference_count }

Device information descriptors.

- enum cl::sycl::info::device_partition_property : int { cl::sycl::info::device_partition_property::unsupported, cl::sycl::info::device_partition_property::partition_equally, cl::sycl::info::device_partition_property::partition_by_counts, cl::sycl::info::device_partition_property::partition_by_affinity_domain, cl::sycl::info::device_partition_property::partition_affinity_domain_next_partitionable }
- enum cl::sycl::info::device_affinity_domain : int { cl::sycl::info::device_affinity_domain::unsupported, cl::sycl::info::device_affinity_domain::numa, cl::sycl::info::device_affinity_domain::L4_cache, cl::sycl::info::device_affinity_domain::L3_cache, cl::sycl::info::device_affinity_domain::L2_cache, cl::sycl::info::device_affinity_domain::next_partitionable }
- enum cl::sycl::info::device_partition_type : int { cl::sycl::info::device_partition_type::no_partition, cl::sycl::info::device_partition_type::numa, cl::sycl::info::device_partition_type::L4_cache, cl::sycl::info::device_partition_type::L3_cache, cl::sycl::info::device_partition_type::L2_cache, cl::sycl::info::device_partition_type::L1_cache }
- enum cl::sycl::info::local_mem_type : int { cl::sycl::info::local_mem_type::none, cl::sycl::info::local_mem_type::local, cl::sycl::info::local_mem_type::global }
- enum cl::sycl::info::fp_config : int { cl::sycl::info::fp_config::denorm, cl::sycl::info::fp_config::inf_nan, cl::sycl::info::fp_config::round_to_nearest, cl::sycl::info::fp_config::round_to_zero, cl::sycl::info::fp_config::round_to_inf, cl::sycl::info::fp_config::fma, cl::sycl::info::fp_config::correctly_rounded_divide_sqrt, cl::sycl::info::fp_config::soft_float }
- enum cl::sycl::info::global_mem_cache_type : int { cl::sycl::info::global_mem_cache_type::none, cl::sycl::info::global_mem_cache_type::read_only, cl::sycl::info::global_mem_cache_type::write_only }
- enum cl::sycl::info::device_execution_capabilities : unsigned int { cl::sycl::info::device_execution_capabilities::exec_kernel, cl::sycl::info::device_execution_capabilities::exec_native_kernel }
- enum cl::sycl::info::platform : unsigned int { cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_PROFILE), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VERSION), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_NAME), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VENDOR), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_EXTENSIONS) }

Platform information descriptors.

Functions

- template<>
  auto cl::sycl::device::get_info< info::device::max_work_group_size > () const
- template<>
  auto cl::sycl::device::get_info< info::device::max_compute_units > () const
- template<>
  auto cl::sycl::device::get_info< info::device::device_type > () const
- template<>
  auto cl::sycl::device::get_info< info::device::local_mem_size > () const
- template<>
  auto cl::sycl::device::get_info< info::device::max_mem_alloc_size > () const
- template<>
  auto cl::sycl::device::get_info< info::device::vendor > () const

- template<>
  auto cl::sycl::device::get_info< info::device::name > () const
- template<>
  auto cl::sycl::device::get_info< info::device::profile > () const
- static vector_class< device > cl::sycl::device::get_devices (info::device_type device_type=info::device_type::all) TRISYCL_WEAK_ATTRIB_SUFFIX

  Return a list of all available devices.
- vector_class< cl::sycl::device > cl::sycl::detail::host_platform::get_devices (info::device_type device_type) const override

  Get all the available devices for the host platform.
- vector_class< cl::sycl::device > cl::sycl::detail::opencl_platform::get_devices (info::device_type device_type) const override

  Get all the available devices for this OpenCL platform.

Variables

- TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_context, detail::opencl_context > opencl_context::cache cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX

8.3.1 Detailed Description

SYCL host context.

Todo The implementation is quite minimal for now. :-)

8.3.2 Class Documentation

8.3.2.1 class cl::sycl::detail::context

Definition at line 24 of file context.hpp.

Inheritance diagram for cl::sycl::detail::context:

Public Member Functions

- virtual cl_context get () const =0

  Return the underlying `cl_context` of the `cl::sycl::context`.
- virtual boost::compute::context & get_boost_compute ()=0

  Return the underlying `boost::compute::context` of the `cl::sycl::context`.
- virtual boost::compute::command_queue & get_boost_queue ()=0

  Return the underlying `boost::compute::command_queue` associated with the context.
- virtual bool is_host () const =0

  Returns true is the context is a SYCL host context.
- virtual cl::sycl::platform get_platform () const =0

  Returns the SYCL platform that the context is initialized for.
- virtual vector_class< cl::sycl::device > get_devices () const =0

  Returns the set of devices that are part of this context.
- virtual ~context ()

  Virtual to call the real destructor.

8.3.2.1.1 Constructor & Destructor Documentation

8.3.2.1.1.1 ~context()

```
virtual cl::sycl::detail::context::~context ( ) [inline], [virtual]
```

Virtual to call the real destructor.

Definition at line 58 of file context.hpp.

```
00058 {}
```

8.3.2.1.2 Member Function Documentation

8.3.2.1.2.1 get()

```
virtual cl_context cl::sycl::detail::context::get ( ) const [pure virtual]
```

Return the underlying `cl_context` of the `cl::sycl::context`.

Implemented in cl::sycl::detail::opencl_context, and cl::sycl::detail::host_context.

8.3.2.1.2.2 get_boost_compute()

```
virtual boost::compute::context& cl::sycl::detail::context::get_boost_compute ( ) [pure virtual]
```

Return the underlying `boost::compute::context` of the `cl::sycl::context`.

Implemented in cl::sycl::detail::opencl_context, and cl::sycl::detail::host_context.

8.3.2.1.2.3 get_boost_queue()

```
virtual boost::compute::command_queue& cl::sycl::detail::context::get_boost_queue ( ) [pure virtual]
```

Return the underlying `boost::compute::command_queue` associated with the context.

Implemented in cl::sycl::detail::opencl_context, and cl::sycl::detail::host_context.

8.3.2.1.2.4 get_devices()

```
virtual vector_class<cl::sycl::device> cl::sycl::detail::context::get_devices ( ) const [pure virtual]
```

Returns the set of devices that are part of this context.

Todo virtual cannot be templated template <info::context Param> typename info::param_traits<info::context, Param>::type get_info() const = 0;

Implemented in cl::sycl::detail::host_context, and cl::sycl::detail::opencl_context.

8.3.2.1.2.5 get_platform()

```
virtual cl::sycl::platform cl::sycl::detail::context::get_platform ( ) const [pure virtual]
```

Returns the SYCL platform that the context is initialized for.

Implemented in cl::sycl::detail::opencl_context, and cl::sycl::detail::host_context.

8.3.2.1.2.6 is_host()

```
virtual bool cl::sycl::detail::context::is_host ( ) const [pure virtual]
```

Returns true is the context is a SYCL host context.

Implemented in cl::sycl::detail::host_context, and cl::sycl::detail::opencl_context.

8.3.2.2 class cl::sycl::detail::host_context

Definition at line 32 of file host_context.hpp.

Inheritance diagram for cl::sycl::detail::host_context:

Collaboration diagram for cl::sycl::detail::host_context:

Public Member Functions

- cl_context get () const override

  Return the underlying `cl_context` of the `cl::sycl::context`.
- boost::compute::context & get_boost_compute () override

  Return the SYCL platform that the context is initialized for.
- boost::compute::command_queue & get_boost_queue () override

  Return the internal OpenCL queue that is associated to the host context.
- bool is_host () const override

  Return true since the context is a SYCL host context.
- cl::sycl::platform get_platform () const override

  Return the platform of the context.
- vector_class< cl::sycl::device > get_devices () const override

  Returns the set of devices that are part of this context.

Additional Inherited Members

8.3.2.2.1 Member Function Documentation

8.3.2.2.1.1 get()

```
cl_context cl::sycl::detail::host_context::get ( ) const  [inline], [override], [virtual]
```

Return the underlying `cl_context` of the `cl::sycl::context`.

This throws an error since there is no OpenCL context associated to the host device.

Implements cl::sycl::detail::context.

Definition at line 43 of file host_context.hpp.

```
00043                                   {
00044     throw non_cl_error("The host context has no OpenCL context");
00045   }
```

8.3.2.2.1.2 get_boost_compute()

```
boost::compute::context& cl::sycl::detail::host_context::get_boost_compute ( )  [inline],
[override], [virtual]
```

Return the SYCL platform that the context is initialized for.

This throws an error since there is no `boost::compute` context associated to the host device.

Implements cl::sycl::detail::context.

Definition at line 53 of file host_context.hpp.

```
00053                                                   {
00054     throw non_cl_error("The host context has no OpenCL context");
00055   }
```

8.3.2.2.1.3 get_boost_queue()

```
boost::compute::command_queue& cl::sycl::detail::host_context::get_boost_queue ( ) [inline], [override], [virtual]
```

Return the internal OpenCL queue that is associated to the host context.

This throws an error since there is no `boost::compute::command_queue` associated to the host context.

Implements cl::sycl::detail::context.

Definition at line 64 of file host_context.hpp.

Referenced by cl::sycl::detail::buffer_base::sync_with_host().

```
00064                                                    {
00065     throw non_cl_error("The host context cannot have an OpenCL queue");
00066   }
```

Here is the caller graph for this function:

8.3.2.2.1.4 get_devices()

```
vector_class<cl::sycl::device> cl::sycl::detail::host_context::get_devices ( ) const [inline], [override], [virtual]
```

Returns the set of devices that are part of this context.

It should only return the host device itself.

Todo To be implemented

Implements cl::sycl::detail::context.

Definition at line 107 of file host_context.hpp.

```
00107                                 {
00108     // Return just the host device
00109     return { {} };
00110   }
```

8.3.2.2.1.5 get_platform()

```
cl::sycl::platform cl::sycl::detail::host_context::get_platform ( ) const [inline], [override], [virtual]
```

Return the platform of the context.

Return synchronous errors via the SYCL exception class.

Implements cl::sycl::detail::context.

Definition at line 80 of file host_context.hpp.

References cl::sycl::detail::unimplemented().

```
00080                                                   {
00081     // Return the host platform
00082     return {};
00083   }
```

Here is the call graph for this function:

8.3.2.2.1.6 is_host()

```
bool cl::sycl::detail::host_context::is_host ( ) const [inline], [override], [virtual]
```

Return true since the context is a SYCL host context.

Implements cl::sycl::detail::context.

Definition at line 71 of file host_context.hpp.

```
00071                                   {
00072     return true;
00073   }
```

8.3.2.3 class cl::sycl::context

SYCL context.

The context class encapsulates an OpenCL context, which is implicitly created and the lifetime of the context instance defines the lifetime of the underlying OpenCL context instance.

On destruction clReleaseContext is called.

The default context is the SYCL host context containing only the SYCL host device.

Todo The implementation is quite minimal for now.

Definition at line 48 of file context.hpp.

Inheritance diagram for cl::sycl::context:

Collaboration diagram for cl::sycl::context:

Public Member Functions

- context (async_handler asyncHandler)

  Constructs a context object for SYCL host using an async_handler for handling asynchronous errors.
- context (cl_context clContext, async_handler asyncHandler=nullptr)

  Make a SYCL context from an OpenCL context.
- context (const boost::compute::context &c, async_handler asyncHandler=nullptr)

  Build a SYCL context from a Boost.Compute context.
- context (const device_selector &deviceSelector, async_handler asyncHandler=nullptr)

  Constructs a context object using a device_selector object.
- context (const device &dev, async_handler asyncHandler=nullptr)

  Constructs a context object using a device object.
- context (const platform &plt, async_handler asyncHandler=nullptr)

  Constructs a context object using a platform object.
- context (const vector_class< device > &deviceList, async_handler asyncHandler=nullptr)
- context ()

  Default constructor that chooses the context according the heuristics of the default selector.
- cl_context get () const

  Return the underlying `cl_context` object, after retaining the `cl_context`.
- boost::compute::context & get_boost_compute () const

  Return the underlying `boost::compute::context` of the `cl::sycl::context`.
- boost::compute::command_queue & get_boost_queue () const

  Return the internal queue that is associated to the context and used by triSYCL to move data between some different contests for example.
- bool is_host () const

  Specifies whether the context is in SYCL Host Execution Mode.
- platform get_platform ()

  Returns the SYCL platform that the context is initialized for.

- vector_class< device > get_devices () const

  Returns the set of devices that are part of this context.
- template<info::context Param>
  info::param_traits< info::context, Param >::type get_info () const

  Queries OpenCL information for the under-lying cl context.

Private Types

- using implementation_t = detail::shared_ptr_implementation< context, detail::context >

Additional Inherited Members

8.3.2.3.1 Member Typedef Documentation

8.3.2.3.1.1 implementation_t

```
using cl::sycl::context::implementation_t = detail::shared_ptr_implementation<context, detail↩
::context>  [private]
```

Definition at line 56 of file context.hpp.

8.3.2.3.2 Constructor & Destructor Documentation

8.3.2.3.2.1 context() [1/8]

```
cl::sycl::context::context (
            async_handler asyncHandler )  [inline], [explicit]
```

Constructs a context object for SYCL host using an async_handler for handling asynchronous errors.

Note that the default case asyncHandler = nullptr is handled by the default constructor.

Definition at line 69 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00069                                                {
00070     detail::unimplemented();
00071   }
```

Here is the call graph for this function:

8.3.2.3.2.2 context() [2/8]

```
cl::sycl::context::context (
            cl_context clContext,
            async_handler asyncHandler = nullptr )  [inline]
```

Make a SYCL context from an OpenCL context.

The constructor executes a retain on the `cl_context`.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the `async_handler`, if provided.

Definition at line 83 of file context.hpp.

```
00084     : context { boost::compute::context { clContext }, asyncHandler } {}
```

8.3.2.3.2.3 context() [3/8]

```
cl::sycl::context::context (
            const boost::compute::context & c,
            async_handler asyncHandler = nullptr )  [inline]
```

Build a SYCL context from a Boost.Compute context.

Definition at line 88 of file context.hpp.

References cl::sycl::detail::opencl_context::instance().

```
00090     : implementation_t { detail::opencl_context::instance(
      c) } {}
```

Here is the call graph for this function:

8.3.2.3.2.4 context() [4/8]

```
cl::sycl::context::context (
            const device_selector & deviceSelector,
            async_handler asyncHandler = nullptr )  [inline]
```

Constructs a context object using a device_selector object.

The context is constructed with a single device retrieved from the device_selector object provided.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩handler, if provided.

Definition at line 101 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00102                                                   {
00103     detail::unimplemented();
00104   }
```

Here is the call graph for this function:

8.3.2.3.2.5 context() [5/8]

```
cl::sycl::context::context (
            const device & dev,
            async_handler asyncHandler = nullptr )  [inline]
```

Constructs a context object using a device object.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_↩handler, if provided.

Definition at line 112 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00113                                                   {
00114     detail::unimplemented();
00115   }
```

Here is the call graph for this function:

8.3.2.3.2.6 context() [6/8]

```
cl::sycl::context::context (
            const platform & plt,
            async_handler asyncHandler = nullptr )  [inline]
```

Constructs a context object using a platform object.

Return synchronous errors via the SYCL exception class and asynchronous errors are handled via the async_handler, if provided.

Definition at line 123 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00124                                                   {
00125     detail::unimplemented();
00126   }
```

Here is the call graph for this function:

8.3.2.3.2.7 context() [7/8]

```
cl::sycl::context::context (
            const vector_class< device > & deviceList,
            async_handler asyncHandler = nullptr )  [inline]
```

Definition at line 137 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00138                                                   {
00139     detail::unimplemented();
00140   }
```

Here is the call graph for this function:

8.3.2.3.2.8 context() [8/8]

```
cl::sycl::context::context ( )  [inline]
```

Default constructor that chooses the context according the heuristics of the default selector.

Return synchronous errors via the SYCL exception class.

Get the default constructors back.

Definition at line 149 of file context.hpp.

References cl::sycl::detail::singleton< host_context >::instance().

```
00149 : implementation_t { detail::host_context::instance() } {}
```

Here is the call graph for this function:

8.3.2.3.3 Member Function Documentation

8.3.2.3.3.1 get()

```
cl_context cl::sycl::context::get ( ) const  [inline]
```

Return the underlying `cl_context` object, after retaining the `cl_context`.

Retains a reference to the returned `cl_context` object.

Caller should release it when finished.

Definition at line 160 of file context.hpp.

References cl::sycl::detail::shared_ptr_implementation< context, detail::context >::implementation.

```
00160                                {
00161     return implementation->get();
00162   }
```

8.3.2.3.3.2 get_boost_compute()

```
boost::compute::context& cl::sycl::context::get_boost_compute ( ) const  [inline]
```

Return the underlying `boost::compute::context` of the `cl::sycl::context`.

Definition at line 167 of file context.hpp.

References cl::sycl::detail::shared_ptr_implementation< context, detail::context >::implementation.

Referenced by cl::sycl::detail::buffer_base::create_in_cache().

```
00167                                                    {
00168     return implementation->get_boost_compute();
00169   }
```

Here is the caller graph for this function:

8.3.2.3.3.3 get_boost_queue()

```
boost::compute::command_queue& cl::sycl::context::get_boost_queue ( ) const  [inline]
```

Return the internal queue that is associated to the context and used by triSYCL to move data between some different contests for example.

Definition at line 175 of file context.hpp.

References cl::sycl::detail::shared_ptr_implementation< context, detail::context >::implementation.

Referenced by cl::sycl::detail::buffer_base::update_buffer_state().

```
00175                                                   {
00176     return implementation->get_boost_queue();
00177   }
```

Here is the caller graph for this function:

8.3.2.3.3.4 get_devices()

```
vector_class<device> cl::sycl::context::get_devices ( ) const  [inline]
```

Returns the set of devices that are part of this context.

Todo To be implemented

Definition at line 198 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00198                                           {
00199     detail::unimplemented();
00200     return {};
00201   }
```

Here is the call graph for this function:

8.3.2.3.3.5 get_info()

```
template<info::context Param>
info::param_traits<info::context, Param>::type cl::sycl::context::get_info ( ) const  [inline]
```

Queries OpenCL information for the under-lying cl context.

Todo To be implemented

Definition at line 209 of file context.hpp.

References cl::sycl::detail::unimplemented().

```
00209                                                                   {
00210     detail::unimplemented();
00211     return {};
00212   }
```

Here is the call graph for this function:

8.3.2.3.3.6 get_platform()

```
platform cl::sycl::context::get_platform ( )
```

Returns the SYCL platform that the context is initialized for.

Todo To be implemented

Referenced by is_host().

Here is the caller graph for this function:

8.3.2.3.3.7 is_host()

```
bool cl::sycl::context::is_host ( ) const  [inline]
```

Specifies whether the context is in SYCL Host Execution Mode.

Definition at line 182 of file context.hpp.

References get_platform(), and cl::sycl::detail::shared_ptr_implementation< context, detail::context >↩ ::implementation.

Referenced by cl::sycl::detail::buffer_base::update_buffer_state().

```
00182                        {
00183     return implementation->is_host();
00184   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.2.4 class cl::sycl::detail::device

An abstract class representing various models of SYCL devices.

Definition at line 25 of file device.hpp.

Inheritance diagram for cl::sycl::detail::device:

Public Member Functions

- virtual cl_device_id get () const =0

  Return the cl_device_id of the underlying OpenCL platform.
- virtual boost::compute::device & get_boost_compute ()=0

  Return the underlying Boost.Compute device, if any.
- virtual bool is_host () const =0

  Return true if the device is a SYCL host device.
- virtual bool is_cpu () const =0

  Return true if the device is an OpenCL CPU device.
- virtual bool is_gpu () const =0

  Return true if the device is an OpenCL GPU device.
- virtual bool is_accelerator () const =0

  Return true if the device is an OpenCL accelerator device.
- virtual cl::sycl::platform get_platform () const =0

  Return the platform of device.
- virtual bool has_extension (const string_class &extension) const =0

  Query the device for OpenCL info::device info.
- virtual ~device ()

8.3.2.4.1 Constructor & Destructor Documentation

8.3.2.4.1.1 ~device()

```
virtual cl::sycl::detail::device::~device ( )  [inline], [virtual]
```

Definition at line 70 of file device.hpp.

```
00070 {}
```

8.3.2.4.2 Member Function Documentation

8.3.2.4.2.1 get()

```
virtual cl_device_id cl::sycl::detail::device::get ( ) const  [pure virtual]
```

Return the cl_device_id of the underlying OpenCL platform.

Implemented in cl::sycl::detail::opencl_device, and cl::sycl::detail::host_device.

8.3.2.4.2.2 get_boost_compute()

```
virtual boost::compute::device& cl::sycl::detail::device::get_boost_compute ( )  [pure virtual]
```

Return the underlying Boost.Compute device, if any.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.4.2.3 get_platform()

```
virtual cl::sycl::platform cl::sycl::detail::device::get_platform ( ) const  [pure virtual]
```

Return the platform of device.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.4.2.4 has_extension()

```
virtual bool cl::sycl::detail::device::has_extension (
            const string_class & extension ) const  [pure virtual]
```

Query the device for OpenCL info::device info.

Todo virtual cannot be templated template <typename t>=""> virtual T get_info(info::device param) const = 0;

Specify whether a specific extension is supported on the device.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.4.2.5 is_accelerator()

```
virtual bool cl::sycl::detail::device::is_accelerator ( ) const  [pure virtual]
```

Return true if the device is an OpenCL accelerator device.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.4.2.6 is_cpu()

```
virtual bool cl::sycl::detail::device::is_cpu ( ) const  [pure virtual]
```

Return true if the device is an OpenCL CPU device.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.4.2.7 is_gpu()

```
virtual bool cl::sycl::detail::device::is_gpu ( ) const  [pure virtual]
```

Return true if the device is an OpenCL GPU device.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.4.2.8 is_host()

```
virtual bool cl::sycl::detail::device::is_host ( ) const  [pure virtual]
```

Return true if the device is a SYCL host device.

Implemented in cl::sycl::detail::host_device, and cl::sycl::detail::opencl_device.

8.3.2.5 class cl::sycl::device

SYCL device.

Definition at line 41 of file device.hpp.

Inheritance diagram for cl::sycl::device:

Collaboration diagram for cl::sycl::device:

Public Member Functions

- device ()

  The default constructor uses the SYCL host device.
- device (cl_device_id device_id)

  Construct a device class instance using cl_device_id of the OpenCL device.
- device (const boost::compute::device &d)

  Construct a device class instance using a boost::compute::device.
- device (const device_selector &ds)

  Construct a device class instance using the device selector provided.
- cl_device_id get () const

  Return the cl_device_id of the underlying OpenCL platform.
- boost::compute::device get_boost_compute () const

  Return the underlying Boost.Compute device if it is an OpenCL device.
- bool is_host () const

  Return true if the device is the SYCL host device.
- bool is_cpu () const

  Return true if the device is an OpenCL CPU device.
- bool is_gpu () const

  Return true if the device is an OpenCL GPU device.
- bool is_accelerator () const

  Return true if the device is an OpenCL accelerator device.
- info::device_type type () const

  Return the device_type of a device.
- platform get_platform () const

  Return the platform of device.
- template<typename T >
  T get_info (info::device param) const

Query the device for OpenCL info::device info.

- template<info::device Param>
  auto get_info () const

  Query the device for OpenCL info::device info.
- bool has_extension (const string_class &extension) const

  Test if a specific extension is supported on the device.

Static Public Member Functions

- static vector_class< device > get_devices (info::device_type device_type=info::device_type::all) TRISYCL↩_WEAK_ATTRIB_SUFFIX

  Return a list of all available devices.

Private Types

- using implementation_t = detail::shared_ptr_implementation< device, detail::device >

Additional Inherited Members

8.3.2.5.1 Member Typedef Documentation

8.3.2.5.1.1 implementation_t

```
using cl::sycl::device::implementation_t = detail::shared_ptr_implementation<device, detail↩
::device>  [private]
```

Definition at line 48 of file device.hpp.

8.3.2.5.2 Constructor & Destructor Documentation

8.3.2.5.2.1 device() [1/4]

```
cl::sycl::device::device ( )  [inline]
```

The default constructor uses the SYCL host device.

Definition at line 56 of file device.hpp.

References cl::sycl::detail::singleton< host_device >::instance().

```
00056 : implementation_t { detail::host_device::instance() } {}
```

Here is the call graph for this function:

8.3.2.5.2.2 device() [2/4]

```
cl::sycl::device::device (
            cl_device_id device_id )  [inline]
```

Construct a device class instance using cl_device_id of the OpenCL device.

Return synchronous errors via the SYCL exception class.

Retain a reference to the OpenCL device and if this device was an OpenCL subdevice the device should be released by the caller when it is no longer needed.

Definition at line 69 of file device.hpp.

```
00070     : device { boost::compute::device { device_id } } {}
```

8.3.2.5.2.3 device() [3/4]

```
cl::sycl::device::device (
            const boost::compute::device & d )  [inline]
```

Construct a device class instance using a boost::compute::device.

This is a triSYCL extension for boost::compute interoperation.

Return synchronous errors via the SYCL exception class.

Definition at line 79 of file device.hpp.

References cl::sycl::detail::opencl_device::instance().

```
00080     : implementation_t { detail::opencl_device::instance(d)
      } {}
```

Here is the call graph for this function:

8.3.2.5.2.4 device() [4/4]

```
cl::sycl::device::device (
            const device_selector & ds )  [inline], [explicit]
```

Construct a device class instance using the device selector provided.

Return errors via C++ exception class.

Todo Make it non-explicit in the specification?

Definition at line 91 of file device.hpp.

References get_devices(), and cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

```
00091                                                 {
00092     auto devices = device::get_devices();
00093     if (devices.empty())
00094       // \todo Put a SYCL exception
00095       throw std::domain_error("No device at all! Internal error...");
00096
00097     /* Find the device with the best score according to the given
00098        device_selector */
00099     auto max = std::max_element(devices.cbegin(), devices.cend(),
00100                                 [&] (const device &d1, const device &d2) {
00101                                   return ds(d1) < ds(d2);
00102                                 });
00103     if (ds(*max) < 0)
00104       // \todo Put a SYCL exception
00105       throw std::domain_error("No device selected because no positive "
00106                               "device_selector score found");
00107
00108     // Create the current device as a shared copy of the selected one
00109     implementation = max->implementation;
00110   }
```

Here is the call graph for this function:

8.3.2.5.3 Member Function Documentation

8.3.2.5.3.1 get()

```
cl_device_id cl::sycl::device::get ( ) const  [inline]
```

Return the cl_device_id of the underlying OpenCL platform.

Return synchronous errors via the SYCL exception class.

Retain a reference to the returned cl_device_id object. Caller should release it when finished.

In the case where this is the SYCL host device it will throw an exception.

Definition at line 124 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

```
00124                                {
00125     return implementation->get();
00126   }
```

8.3.2.5.3.2 get_boost_compute()

```
boost::compute::device cl::sycl::device::get_boost_compute ( ) const  [inline]
```

Return the underlying Boost.Compute device if it is an OpenCL device.

This is a triSYCL extension

Definition at line 134 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by cl::sycl::detail::opencl_queue::instance().

```
00134                                                {
00135     return implementation->get_boost_compute();
00136   }
```

Here is the caller graph for this function:

8.3.2.5.3.3 get_info() [1/2]

```
template<typename T >
T cl::sycl::device::get_info (
            info::device param ) const  [inline]
```

Query the device for OpenCL info::device info.

Return synchronous errors via the SYCL exception class.

Todo

Definition at line 211 of file device.hpp.

References get_info().

```
00211                                   {
00212     //return implementation->get_info<Param>(param);
00213   }
```

Here is the call graph for this function:

8.3.2.5.3.4 get_info() [2/2]

```
template<info::device Param>
auto cl::sycl::device::get_info ( ) const  [inline]
```

Query the device for OpenCL info::device info.

Return synchronous errors via the SYCL exception class.

Todo

Referenced by get_info().

Here is the caller graph for this function:

8.3.2.5.3.5 get_platform()

```
platform cl::sycl::device::get_platform ( ) const  [inline]
```

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Definition at line 188 of file device.hpp.

References cl::sycl::info::all, get_devices(), cl::sycl::detail::shared_ptr_implementation< device, detail::device >↩ ::implementation, and TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00188                                {
00189     return implementation->get_platform();
00190   }
```

Here is the call graph for this function:

8.3.2.5.3.6 has_extension()

```
bool cl::sycl::device::has_extension (
            const string_class & extension ) const  [inline]
```

Test if a specific extension is supported on the device.

Definition at line 233 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

```
00233                                                    {
00234     return implementation->has_extension(extension);
00235   }
```

8.3.2.5.3.7 is_accelerator()

```
bool cl::sycl::device::is_accelerator ( ) const  [inline]
```

Return true if the device is an OpenCL accelerator device.

Definition at line 159 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by type().

```
00159                                {
00160     return implementation->is_accelerator();
00161   }
```

Here is the caller graph for this function:

8.3.2.5.3.8 is_cpu()

```
bool cl::sycl::device::is_cpu ( ) const  [inline]
```

Return true if the device is an OpenCL CPU device.

Definition at line 147 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by type().

```
00147                        {
00148     return implementation->is_cpu();
00149   }
```

Here is the caller graph for this function:

8.3.2.5.3.9 is_gpu()

```
bool cl::sycl::device::is_gpu ( ) const  [inline]
```

Return true if the device is an OpenCL GPU device.

Definition at line 153 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by type().

```
00153                        {
00154     return implementation->is_gpu();
00155   }
```

Here is the caller graph for this function:

8.3.2.5.3.10 is_host()

```
bool cl::sycl::device::is_host ( ) const  [inline]
```

Return true if the device is the SYCL host device.

Definition at line 141 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< device, detail::device >::implementation.

Referenced by cl::sycl::device_type_selector::operator()(), cl::sycl::queue::queue(), and type().

```
00141                        {
00142     return implementation->is_host();
00143   }
```

Here is the caller graph for this function:

8.3.2.5.3.11 type()

```
info::device_type cl::sycl::device::type ( ) const  [inline]
```

Return the device_type of a device.

Todo Present in Boost.Compute, to be added to the specification

Definition at line 169 of file device.hpp.

References cl::sycl::info::accelerator, cl::sycl::info::cpu, cl::sycl::info::gpu, cl::sycl::info::host, is_accelerator(), is_cpu(), is_gpu(), and is_host().

Referenced by cl::sycl::device_type_selector::operator()().

```
00169                                {
00170     if (is_host())
00171       return info::device_type::host;
00172     else if (is_cpu())
00173       return info::device_type::cpu;
00174     else if (is_gpu())
00175       return info::device_type::gpu;
00176     else if (is_accelerator())
00177       return info::device_type::accelerator;
00178     else
00179       // \todo Put a SYCL exception
00180       throw std::domain_error("Unknown cl::sycl::info::device_type");
00181   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.2.6 class cl::sycl::device_type_selector

A device selector by device_type.

Todo To be added to the specification

Definition at line 32 of file device_selector_tail.hpp.

Inheritance diagram for cl::sycl::device_type_selector:

Collaboration diagram for cl::sycl::device_type_selector:

Public Member Functions

- device_type_selector (info::device_type device_type)
- int operator() (const device &dev) const override

  This pure virtual operator allows the customization of device selection.

Private Attributes

- info::device_type device_type

  The device_type to select.
- device default_device

  Cache the default device to select with the default device selector.

8.3.2.6.1 Constructor & Destructor Documentation

8.3.2.6.1.1 device_type_selector()

```
cl::sycl::device_type_selector::device_type_selector (
            info::device_type device_type )  [inline]
```

Definition at line 48 of file device_selector_tail.hpp.

References cl::sycl::info::automatic.

```
00049     : device_type { device_type } {
00050     // The default device selection heuristic
00051 #ifdef TRISYCL_OPENCL
00052     if (device_type == info::device_type::automatic) {
00053       // Ask Boost.Compute for the default OpenCL device
00054       try {
00055         default_device = boost::compute::system::default_device();
00056       }
00057       catch (...) {
00058         /* If there is no OpenCL device, just keep the
00059            default-constructed device, which is the host device */
00060       }
00061     }
00062 #endif
00063   }
```

8.3.2.6.2 Member Function Documentation

8.3.2.6.2.1 operator()()

```
int cl::sycl::device_type_selector::operator() (
            const device & dev ) const  [inline], [override], [virtual]
```

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implements cl::sycl::device_selector.

Definition at line 67 of file device_selector_tail.hpp.

References cl::sycl::info::all, cl::sycl::info::automatic, cl::sycl::device::is_host(), cl::sycl::info::opencl, and cl::sycl↩::device::type().

```
00067                                                   {
00068     if (device_type == info::device_type::all)
00069       // All devices fit all
00070       return 1;
00071
00072     if (device_type == info::device_type::automatic)
00073       // Only select the default device
00074       return dev == default_device ? 1 : -1;
00075
00076     if (device_type == info::device_type::opencl)
00077       // For now, any non host device is an OpenCL device
00078       return dev.is_host() ? -1 : 1;
00079
00080     return dev.type() == device_type ? 1 : -1;
00081   }
```

Here is the call graph for this function:

8.3.2.6.3 Member Data Documentation

8.3.2.6.3.1 default_device

```
device cl::sycl::device_type_selector::default_device  [private]
```

Cache the default device to select with the default device selector.

This is the host device at construction time and remains as is if there is no openCL device

Definition at line 44 of file device_selector_tail.hpp.

8.3.2.6.3.2 device_type

```
info::device_type cl::sycl::device_type_selector::device_type [private]
```

The device_type to select.

Definition at line 37 of file device_selector_tail.hpp.

8.3.2.7 class cl::sycl::device_typename_selector

template<info::device_type DeviceType>
class cl::sycl::device_typename_selector< DeviceType >

Select a device by template device_type parameter.

Todo To be added to the specification

Definition at line 91 of file device_selector_tail.hpp.

Inheritance diagram for cl::sycl::device_typename_selector< DeviceType >:

Collaboration diagram for cl::sycl::device_typename_selector< DeviceType >:

Public Member Functions

- device_typename_selector ()

8.3.2.7.1 Constructor & Destructor Documentation

8.3.2.7.1.1 device_typename_selector()

```
template<info::device_type DeviceType>
cl::sycl::device_typename_selector< DeviceType >::device_typename_selector ( ) [inline]
```

Definition at line 95 of file device_selector_tail.hpp.

```
00095 : device_type_selector { DeviceType } {}
```

8.3.2.8 class cl::sycl::device_selector

The SYCL heuristics to select a device.

The device with the highest score is selected

Definition at line 26 of file device_selector.hpp.

Inheritance diagram for cl::sycl::device_selector:

Public Member Functions

- void select_device () const

  Returns a selected device using the functor operator defined in sub-classes operator()(const device &dev)
- virtual int operator() (const device &dev) const =0

  This pure virtual operator allows the customization of device selection.
- virtual ~device_selector ()

  Virtual destructor so the final destructor can be called if any.

8.3.2.8.1 Constructor & Destructor Documentation

8.3.2.8.1.1 ~device_selector()

```
virtual cl::sycl::device_selector::~device_selector ( )  [inline], [virtual]
```

Virtual destructor so the final destructor can be called if any.

Definition at line 52 of file device_selector.hpp.

```
00052 {}
```

8.3.2.8.2 Member Function Documentation

8.3.2.8.2.1 operator()()

```
virtual int cl::sycl::device_selector::operator() (
            const device & dev ) const  [pure virtual]
```

This pure virtual operator allows the customization of device selection.

It defines the behavior of the device_selector functor called by the SYCL runtime on device selection. It returns a "score" for each device in the system and the highest rated device will be used by the SYCL runtime.

Implemented in cl::sycl::device_type_selector.

Referenced by select_device().

Here is the caller graph for this function:

8.3.2.8.2.2 select_device()

```
void cl::sycl::device_selector::select_device ( ) const  [inline]
```

Returns a selected device using the functor operator defined in sub-classes operator()(const device &dev)

Todo Remove this from specification

Definition at line 35 of file device_selector.hpp.

References operator()().

```
00035                                   {
00036     //    return {};
00037   }
```

Here is the call graph for this function:

8.3.2.9 class cl::sycl::handler

Command group handler class.

A command group handler object can only be constructed by the SYCL runtime.

All of the accessors defined in the command group scope take as a parameter an instance of the command group handler and all the kernel invocation functions are methods of this class.

Definition at line 46 of file handler.hpp.

Inheritance diagram for cl::sycl::handler:

Collaboration diagram for cl::sycl::handler:

Public Member Functions

- handler (const std::shared_ptr< detail::queue > &q)
- template<typename DataType , int Dimensions, access::mode Mode, access::target Target = access::target::global_buffer>
  void set_arg (int arg_index, accessor< DataType, Dimensions, Mode, Target > &&acc_obj)

  Set accessor kernel arg for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.
- template<typename T , typename = std::enable_if_t<is_wrapper<T>::value>>
  void set_arg (int arg_index, T &&scalar_value)

Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface with a wrapper type.

- template<typename T >
  std::enable_if_t<!is_wrapper< T >::value > set_arg (int arg_index, T &&scalar_value)

  Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface without a wrapper type.

- template<typename... Ts>
  void set_args (Ts &&... args)

  Set all kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

- template<typename KernelName = std::nullptr_t, typename ParallelForFunctor >
  void single_task (ParallelForFunctor &&f)

  Kernel invocation method of a kernel defined as a lambda or functor.

- TRISYCL_parallel_for_functor_GLOBAL (1) TRISYCL_parallel_for_functor_GLOBAL(2) TRISYCL_parallel↩_for_functor_GLOBAL(3) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1) TRISYCL_ParallelFor↩Functor_GLOBAL_OFFSET(2) TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3) template< typename KernelName

  Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and offset and given an id or item for indexing in the indexing space defined by range.

- int ParallelForFunctor void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)
- template<typename KernelName = std::nullptr_t, int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_group (nd_range< Dimensions > r, ParallelForFunctor f)

  Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

- template<typename KernelName = std::nullptr_t, int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_group (range< Dimensions > r1, range< Dimensions > r2, ParallelForFunctor f)

  Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

- void single_task (kernel sycl_kernel)

  Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.

Public Attributes

- std::shared_ptr< detail::task > task

  Attach the task and accessors to it.

- int Dimensions

Private Member Functions

- template<std::size_t... Is, typename... Ts>
  void dispatch_set_arg (std::index_sequence< Is... >, Ts &&... args)

  Helper to individually call set_arg() for each argument.

- template<typename KernelName , typename Kernel >
  void schedule_kernel (Kernel k)

  Schedule the kernel.

8.3.2.9.1 Constructor & Destructor Documentation

8.3.2.9.1.1 handler()

```
cl::sycl::handler::handler (
            const std::shared_ptr< detail::queue > & q )  [inline]
```

Definition at line 64 of file handler.hpp.

References Dimensions, and cl::sycl::access::global_buffer.

```
00064                                                 {
00065     // Create a new task for this command_group
00066     task = std::make_shared<detail::task>(q);
00067   }
```

8.3.2.9.2 Member Function Documentation

8.3.2.9.2.1 dispatch_set_arg()

```
template<std::size_t...  Is, typename...  Ts>
void cl::sycl::handler::dispatch_set_arg (
            std::index_sequence< Is...  > ,
            Ts &&...  args )  [inline], [private]
```

Helper to individually call set_arg() for each argument.

Definition at line 132 of file handler.hpp.

References set_arg().

Referenced by set_args().

```
00132                                                                 {
00133     // Use an intermediate tuple to ease individual argument access
00134     auto &&t = std::make_tuple(std::forward<Ts>(args)...);
00135     // Dispatch individual set_arg() for each argument
00136     auto just_to_evaluate = {
00137       0 /*< At least 1 element to deal with empty set_args() */,
00138       ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
00139     };
00140     // Remove the warning about unused variable
00141     static_cast<void>(just_to_evaluate);
00142   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.2.9.2.2 parallel_for()

```
int ParallelForFunctor void cl::sycl::handler::parallel_for (
            nd_range< Dimensions > r,
            ParallelForFunctor f )  [inline]
```

Definition at line 315 of file handler.hpp.

References cl::sycl::detail::parallel_for().

```
00316                                          {
00317     schedule_kernel<KernelName>([=] { detail::parallel_for(r, f); });
00318   }
```

Here is the call graph for this function:

8.3.2.9.2.3 parallel_for_work_group() [1/2]

```
template<typename KernelName = std::nullptr_t, int Dimensions = 1, typename ParallelForFunctor
>
void cl::sycl::handler::parallel_for_work_group (
            nd_range< Dimensions > r,
            ParallelForFunctor f )  [inline]
```

Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

May contain multiple kernel built-in parallel_for_work_item functions representing the execution on each work-item.

Launch num_work_groups work-groups of runtime-defined size. Described in detail in 3.5.3.

Parameters

r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 345 of file handler.hpp.

References cl::sycl::detail::parallel_for_workgroup().

Referenced by parallel_for_work_group().

```
00346                                                   {
00347     schedule_kernel<KernelName>([=] {
00348         detail::parallel_for_workgroup(r, f);
00349       });
00350   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.2.9.2.4 parallel_for_work_group() [2/2]

```
template<typename KernelName = std::nullptr_t, int Dimensions = 1, typename ParallelForFunctor
>
void cl::sycl::handler::parallel_for_work_group (
            range< Dimensions > r1,
            range< Dimensions > r2,
            ParallelForFunctor f )  [inline]
```

Hierarchical kernel invocation method of a kernel defined as a lambda encoding the body of each work-group to launch.

May contain multiple kernel built-in parallel_for_work_item functions representing the execution on each work-item.

Launch num_work_groups work-groups of runtime-defined size. Described in detail in 3.5.3.

Parameters

r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 377 of file handler.hpp.

References parallel_for_work_group().

```
00378                                                   {
00379     parallel_for_work_group(nd_range<Dimensions> { r1, r2 }, f);
00380   }
```

Here is the call graph for this function:

8.3.2.9.2.5 schedule_kernel()

```
template<typename KernelName , typename Kernel >
void cl::sycl::handler::schedule_kernel (
            Kernel k )  [inline], [private]
```

Schedule the kernel.

Add a traced version of the kernel in host mode or add the kernel in an instantiating function for later extraction by the compiler

Definition at line 171 of file handler.hpp.

References TRISYCL_DUMP_T.

```
00171                                   {
00172     /* Explicitly capture task by copy instead of having this captured
00173        by reference and task by reference by side effect */
00174     task->schedule(detail::trace_kernel<KernelName>([=, t = task] () mutable {
00175           if (t->owner_queue->is_host())
00176             k();
00177           else {
00178             TRISYCL_DUMP_T("schedule_kernel &k = " << (void *) &k);
00179
00180             // Structure the kernel code so it can be outlined and called
00181             detail::launch_device_kernel<KernelName>(*t, k);
00182             // \todo for now only deal with 1 physical work-item only
00183             t->get_kernel().single_task(t, t->get_queue());
00184           }
00185         }));
00186   }
```

8.3.2.9.2.6 set_arg() [1/3]

```
template<typename DataType , int Dimensions, access::mode Mode, access::target Target = access↩
::target::global_buffer>
void cl::sycl::handler::set_arg (
            int arg_index,
            accessor< DataType, Dimensions, Mode, Target > && acc_obj )  [inline]
```

Set accessor kernel arg for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

The index value specifies which parameter of the OpenCL kernel is being set and the accessor object, which OpenCL buffer or image is going to be given as kernel argument.

Todo Update the specification to use a ref && to the accessor instead?

Todo It is not that clean to have set_arg() associated to a command handler. Rethink the specification?

Todo It seems more logical to have these methods on kernel instead

Definition at line 89 of file handler.hpp.

Referenced by dispatch_set_arg().

```
00090                                                                    {
00091     /* Think about setting the kernel argument before actually calling
00092        the kernel.
00093
00094        Explicitly capture task by copy instead of having this captured
00095        by reference and task by reference by side effect */
00096     task->add_prelude([=, task = task] {
00097         task->set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
00098       });
00099   }
```

Here is the caller graph for this function:

8.3.2.9.2.7 set_arg() [2/3]

```
template<typename T , typename = std::enable_if_t<is_wrapper<T>::value>>
void cl::sycl::handler::set_arg (
            int arg_index,
            T && scalar_value )  [inline]
```

Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface with a wrapper type.

Definition at line 106 of file handler.hpp.

```
00106                                                   {
00107     /* Explicitly capture task by copy instead of having this captured
00108        by reference and task by reference by side effect */
00109     task->add_prelude([=, task = task] {
00110         task->set_arg(arg_index, scalar_value.unwrap());
00111       });
00112   }
```

8.3.2.9.2.8 set_arg() [3/3]

```
template<typename T >
std::enable_if_t<!is_wrapper<T>::value> cl::sycl::handler::set_arg (
            int arg_index,
            T && scalar_value )  [inline]
```

Set kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interoperability interface without a wrapper type.

Definition at line 120 of file handler.hpp.

```
00120                                                 {
00121     /* Explicitly capture task by copy instead of having this captured
00122        by reference and task by reference by side effect */
00123     task->add_prelude([=, task = task] {
00124         task->set_arg(arg_index, scalar_value);
00125       });
00126   }
```

8.3.2.9.2.9 set_args()

```
template<typename...  Ts>
void cl::sycl::handler::set_args (
            Ts &&...  args )  [inline]
```

Set all kernel args for an OpenCL kernel which is used through the SYCL/OpenCL interop interface.

Todo Update the specification to add this function according to https://cvs.khronos.org/bugzilla/show↩_bug.cgi?id=15978 proposal

Definition at line 153 of file handler.hpp.

References dispatch_set_arg().

```
00153                                {
00154     /* Construct a set of increasing argument index to be able to call
00155        the real set_arg */
00156     dispatch_set_arg(std::index_sequence_for<Ts...>{},
00157                      std::forward<Ts>(args)...);
00158   }
```

Here is the call graph for this function:

8.3.2.9.2.10 single_task() [1/2]

```
template<typename KernelName = std::nullptr_t, typename ParallelForFunctor >
void cl::sycl::handler::single_task (
            ParallelForFunctor && f )  [inline]
```

Kernel invocation method of a kernel defined as a lambda or functor.

If it is a lambda function or the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in 3.5.3

SYCL single_task launches a computation without parallelism at launch time.

Parameters

F	specify the kernel to be launched as a single_task
KernelName	is a class type that defines the name to be used for the underlying kernel

Definition at line 205 of file handler.hpp.

References TRISYCL_DUMP_T.

```
00205                                        {
00206     TRISYCL_DUMP_T("single_task &f = " << (void *) &f);
00207
00208     schedule_kernel<KernelName>(std::forward<ParallelForFunctor>(f));
00209   }
```

8.3.2.9.2.11 single_task() [2/2]

```
void cl::sycl::handler::single_task (
            kernel sycl_kernel )  [inline]
```

Kernel invocation method of a kernel defined as pointer to a kernel object, described in detail in 3.5.3.

Todo Add in the spec a version taking a kernel and a functor, to have host fall-back

Definition at line 389 of file handler.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::implementation.

```
00389                                       {
00390     /* For now just use the usual host task system to schedule          \
00391        manually the OpenCL kernels instead of using OpenCL event-based  \
00392        scheduling                                                       \
00393                                                                         \
00394        \todo Move the tracing inside the kernel implementation          \
00395                                                                         \
00396        \todo Simplify this 2 step ugly interface                        \
00397     */                                                                  \
00398     task->set_kernel(sycl_kernel.implementation);                       \
00399     task->schedule(detail::trace_kernel<kernel>([=, t = task] {         \
00400           sycl_kernel.implementation->single_task(t, t->get_queue());   \
00401         }));                                                            \
00402   }
```

8.3.2.9.2.12 TRISYCL_parallel_for_functor_GLOBAL()

```
cl::sycl::handler::TRISYCL_parallel_for_functor_GLOBAL (
            1  )
```

Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and offset and given an id or item for indexing in the indexing space defined by range.

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

global_size	is the global size of the range<>
offset	is the offset to be add to the id<> during iteration
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

Unfortunately, to have implicit conversion to work on the range, the function can not be templated, so instantiate it for all the dimensionsKernel invocation method of a kernel defined as a lambda or functor, for the specified nd_range and given an nd_item for indexing in the indexing space defined by the nd_range

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

r	defines the iteration space with the work-group layout and offset
Dimensions	dimensionality of the iteration space
f	is the kernel functor to execute
ParallelForFunctor	is the kernel functor type
KernelName	is a class type that defines the name to be used for the underlying kernel

8.3.2.9.3 Member Data Documentation

8.3.2.9.3.1 Dimensions

```
int cl::sycl::handler::Dimensions
```

Definition at line 313 of file handler.hpp.

Referenced by handler().

8.3.2.9.3.2 task

```
std::shared_ptr<detail::task> cl::sycl::handler::task
```

Attach the task and accessors to it.

Definition at line 52 of file handler.hpp.

Referenced by cl::sycl::detail::add_buffer_to_task().

8.3.2.10 class cl::sycl::detail::kernel

Abstract SYCL kernel.

Definition at line 31 of file kernel.hpp.

Inheritance diagram for cl::sycl::detail::kernel:

Collaboration diagram for cl::sycl::detail::kernel:

Public Member Functions

- virtual cl_kernel get () const =0

  Return the OpenCL kernel object for this kernel.
- virtual boost::compute::kernel & get_boost_compute ()=0

  Return the Boost.Compute OpenCL kernel object for this kernel.
- virtual void single_task (std::shared_ptr< detail::task > task, std::shared_ptr< detail::queue > q)=0

  Launch a single task of the kernel.
- TRISYCL_ParallelForKernel_RANGE (1) TRISYCL_ParallelForKernel_RANGE(2) TRISYCL_ParallelFor↩Kernel_RANGE(3) virtual ~kernel()

  Return the context that this kernel is defined for.

8.3.2.10.1 Member Function Documentation

8.3.2.10.1.1 get()

```
virtual cl_kernel cl::sycl::detail::kernel::get ( ) const  [pure virtual]
```

Return the OpenCL kernel object for this kernel.

Retains a reference to the returned cl_kernel object. Caller should release it when finished.

Implemented in cl::sycl::detail::opencl_kernel.

8.3.2.10.1.2 get_boost_compute()

```
virtual boost::compute::kernel& cl::sycl::detail::kernel::get_boost_compute ( )  [pure virtual]
```

Return the Boost.Compute OpenCL kernel object for this kernel.

This is an extension.

Implemented in cl::sycl::detail::opencl_kernel.

Referenced by cl::sycl::detail::task::set_arg().

Here is the caller graph for this function:

8.3.2.10.1.3 single_task()

```
virtual void cl::sycl::detail::kernel::single_task (
            std::shared_ptr< detail::task > task,
            std::shared_ptr< detail::queue > q )  [pure virtual]
```

Launch a single task of the kernel.

Implemented in cl::sycl::detail::opencl_kernel.

8.3.2.10.1.4 TRISYCL_ParallelForKernel_RANGE()

```
cl::sycl::detail::kernel::TRISYCL_ParallelForKernel_RANGE (
            1  )  [inline]
```

Return the context that this kernel is defined for.

Return the program that this kernel is part of

Definition at line 67 of file kernel.hpp.

```
00080                   {}
```

8.3.2.11 class cl::sycl::kernel

SYCL kernel.

Todo To be implemented

Todo Check specification

Definition at line 38 of file kernel.hpp.

Inheritance diagram for cl::sycl::kernel:

Collaboration diagram for cl::sycl::kernel:

Public Member Functions

- kernel ()=delete

  The default object is not valid because there is no program or.
- kernel (cl_kernel k)

  Constructor for SYCL kernel class given an OpenCL kernel object with set arguments, valid for enqueuing.
- kernel (const boost::compute::kernel &k)

  Construct a kernel class instance using a boost::compute::kernel.
- cl_kernel get () const

  Return the OpenCL kernel object for this kernel.

Private Types

- using implementation_t = typename kernel::shared_ptr_implementation

Private Attributes

- friend implementation_t

Friends

- class handler

Additional Inherited Members

8.3.2.11.1 Member Typedef Documentation

8.3.2.11.1.1 implementation_t

```
using cl::sycl::kernel::implementation_t = typename kernel::shared_ptr_implementation  [private]
```

Definition at line 44 of file kernel.hpp.

8.3.2.11.2 Constructor & Destructor Documentation

8.3.2.11.2.1 kernel() [1/3]

```
cl::sycl::kernel::kernel ( )  [delete]
```

The default object is not valid because there is no program or.

```
cl_kernel
```

associated with it

8.3.2.11.2.2 kernel() [2/3]

```
cl::sycl::kernel::kernel (
            cl_kernel k )  [inline]
```

Constructor for SYCL kernel class given an OpenCL kernel object with set arguments, valid for enqueuing.

Retains a reference to the `cl_kernel` object. The Caller should release the passed cl_kernel object when it is no longer needed.

Definition at line 69 of file kernel.hpp.

```
00069 : kernel { boost::compute::kernel { k } } {}
```

8.3.2.11.2.3 kernel() [3/3]

```
cl::sycl::kernel::kernel (
            const boost::compute::kernel & k )  [inline]
```

Construct a kernel class instance using a boost::compute::kernel.

This is a triSYCL extension for boost::compute interoperation.

Return synchronous errors via the SYCL exception class.

Definition at line 78 of file kernel.hpp.

References cl::sycl::detail::opencl_kernel::instance().

```
00079     : implementation_t { detail::opencl_kernel::instance(k)
      } {}
```

Here is the call graph for this function:

8.3.2.11.3 Member Function Documentation

8.3.2.11.3.1 get()

```
cl_kernel cl::sycl::kernel::get ( ) const  [inline]
```

Return the OpenCL kernel object for this kernel.

Retains a reference to the returned cl_kernel object. Caller should release it when finished.

Definition at line 87 of file kernel.hpp.

References cl::sycl::detail::shared_ptr_implementation< kernel, detail::kernel >::implementation, and cl::sycl::detail::unimplemented().

```
00087                          {
00088     return implementation->get();
00089   }
```

Here is the call graph for this function:

8.3.2.11.4 Friends And Related Function Documentation

8.3.2.11.4.1 handler

```
friend class handler  [friend]
```

Definition at line 47 of file kernel.hpp.

8.3.2.11.5 Member Data Documentation

8.3.2.11.5.1 implementation_t

```
friend cl::sycl::kernel::implementation_t  [private]
```

Definition at line 50 of file kernel.hpp.

8.3.2.12 class cl::sycl::detail::host_platform

SYCL host platform.

Definition at line 31 of file host_platform.hpp.

Inheritance diagram for cl::sycl::detail::host_platform:

Collaboration diagram for cl::sycl::detail::host_platform:

Public Member Functions

- cl_platform_id get () const override

  Return the cl_platform_id of the underlying OpenCL platform.
- boost::compute::platform & get_boost_compute () const override

  Return the underlying Boost.Compute platform.
- bool is_host () const override

  Return true since this platform is the SYCL host platform.
- string_class get_info_string (info::platform param) const override

  Returning the information parameters for the host platform implementation.
- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the platform.
- vector_class< cl::sycl::device > get_devices (info::device_type device_type) const override

  Get all the available devices for the host platform.

Static Private Attributes

- static auto constexpr platform_extensions = "Xilinx_blocking_pipes"

Additional Inherited Members

8.3.2.12.1 Member Function Documentation

8.3.2.12.1.1 get()

```
cl_platform_id cl::sycl::detail::host_platform::get ( ) const  [inline], [override], [virtual]
```

Return the cl_platform_id of the underlying OpenCL platform.

This throws an error since there is no OpenCL platform associated to the host platform.

Implements cl::sycl::detail::platform.

Definition at line 45 of file host_platform.hpp.

```
00045                                       {
00046     throw non_cl_error("The host platform has no OpenCL platform");
00047   }
```

8.3.2.12.1.2 get_boost_compute()

```
boost::compute::platform& cl::sycl::detail::host_platform::get_boost_compute ( ) const  [inline],
[override], [virtual]
```

Return the underlying Boost.Compute platform.

This throws an error since there is no Boost Compute platform associated to the host platform.

Implements cl::sycl::detail::platform.

Definition at line 55 of file host_platform.hpp.

```
00055                                                                 {
00056     throw
00057       non_cl_error("The host device has no underlying Boost Compute platform");
00058   }
```

8.3.2.12.1.3 get_info_string()

```
string_class cl::sycl::detail::host_platform::get_info_string (
            info::platform param ) const  [inline], [override], [virtual]
```

Returning the information parameters for the host platform implementation.

Implements cl::sycl::detail::platform.

Definition at line 71 of file host_platform.hpp.

References cl::sycl::info::extensions, cl::sycl::info::name, platform_extensions, cl::sycl::info::profile, and cl::sycl↩::info::vendor.

```
00071                                                                   {
00072     switch (param) {
00073     case info::platform::profile:
00074       /* Well... Is the host platform really a full profile whereas it
00075          is not really OpenCL? */
00076       return "FULL_PROFILE";
00077
00078     case info::platform::version:
00079       // \todo I guess it should include the software version too...
00080       return "2.2";
00081
00082     case info::platform::name:
00083       return "triSYCL host platform";
00084
00085     case info::platform::vendor:
00086       return "triSYCL Open Source project";
00087
00088     case info::platform::extensions:
00089       return platform_extensions;
00090
00091     default:
00092       // \todo Define some SYCL exception type for this type of errors
00093       throw std::invalid_argument {
00094         "Unknown parameter value for SYCL platform information" };
00095     }
00096   }
```

8.3.2.12.1.4 has_extension()

```
bool cl::sycl::detail::host_platform::has_extension (
            const string_class & extension ) const  [inline], [override], [virtual]
```

Specify whether a specific extension is supported on the platform.

Todo To be implemented

Implements cl::sycl::detail::platform.

Definition at line 103 of file host_platform.hpp.

References get_devices(), and cl::sycl::detail::unimplemented().

```
00103                                                                   {
00104     detail::unimplemented();
00105     return {};
00106   }
```

Here is the call graph for this function:

8.3.2.12.1.5 is_host()

```
bool cl::sycl::detail::host_platform::is_host ( ) const  [inline], [override], [virtual]
```

Return true since this platform is the SYCL host platform.

Implements cl::sycl::detail::platform.

Definition at line 63 of file host_platform.hpp.

```
00063                                   {
00064     return true;
00065   }
```

8.3.2.12.2 Member Data Documentation

8.3.2.12.2.1 platform_extensions

```
auto constexpr cl::sycl::detail::host_platform::platform_extensions = "Xilinx_blocking_pipes"
[static], [private]
```

Definition at line 35 of file host_platform.hpp.

Referenced by get_info_string().

8.3.2.13 class cl::sycl::detail::opencl_platform

SYCL OpenCL platform.

Definition at line 36 of file opencl_platform.hpp.

Inheritance diagram for cl::sycl::detail::opencl_platform:

Collaboration diagram for cl::sycl::detail::opencl_platform:

Public Member Functions

- cl_platform_id get () const override

  Return the cl_platform_id of the underlying OpenCL platform.
- const boost::compute::platform & get_boost_compute () const override

  Return the underlying Boost.Compute platform.
- bool is_host () const override

  Return false since an OpenCL platform is not the SYCL host platform.
- string_class get_info_string (info::platform param) const override

  Returning the information string parameters for the OpenCL platform.
- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the platform.
- vector_class< cl::sycl::device > get_devices (info::device_type device_type) const override

  Get all the available devices for this OpenCL platform.
- ~opencl_platform () override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_platform > instance (const boost::compute::platform &p)

Private Member Functions

- opencl_platform (const boost::compute::platform &p)

  Only the instance factory can built it.

Private Attributes

- boost::compute::platform p

  Use the Boost Compute abstraction of the OpenCL platform.

Static Private Attributes

- static detail::cache< cl_platform_id, detail::opencl_platform > cache

  A cache to always return the same live platform for a given OpenCL platform.

8.3.2.13.1 Constructor & Destructor Documentation

8.3.2.13.1.1 opencl_platform()

```
cl::sycl::detail::opencl_platform::opencl_platform (
            const boost::compute::platform & p )  [inline], [private]
```

Only the instance factory can built it.

Definition at line 105 of file opencl_platform.hpp.

```
00105 : p { p } {}
```

8.3.2.13.1.2 ~opencl_platform()

```
cl::sycl::detail::opencl_platform::~opencl_platform ( )  [inline], [override]
```

Unregister from the cache on destruction.

Definition at line 110 of file opencl_platform.hpp.

References cache, cl::sycl::detail::cache< Key, Value >::remove(), TRISYCL_WEAK_ATTRIB_PREFIX, and cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00110                                {
00111     cache.remove(p.id());
00112   }
```

Here is the call graph for this function:

8.3.2.13.2 Member Function Documentation

8.3.2.13.2.1 get()

```
cl_platform_id cl::sycl::detail::opencl_platform::get ( ) const  [inline], [override], [virtual]
```

Return the cl_platform_id of the underlying OpenCL platform.

Implements cl::sycl::detail::platform.

Definition at line 51 of file opencl_platform.hpp.

```
00051                                       {
00052     return p.id();
00053   }
```

8.3.2.13.2.2 get_boost_compute()

```
const boost::compute::platform& cl::sycl::detail::opencl_platform::get_boost_compute ( ) const
[inline], [override], [virtual]
```

Return the underlying Boost.Compute platform.

Implements cl::sycl::detail::platform.

Definition at line 57 of file opencl_platform.hpp.

References p.

Referenced by get_devices().

```
00057                                                                   {
00058     return p;
00059   }
```

Here is the caller graph for this function:

8.3.2.13.2.3 get_info_string()

```
string_class cl::sycl::detail::opencl_platform::get_info_string (
            info::platform param ) const  [inline], [override], [virtual]
```

Returning the information string parameters for the OpenCL platform.

Implements cl::sycl::detail::platform.

Definition at line 69 of file opencl_platform.hpp.

```
00069                                                                   {
00070     /* Use the fact that the triSYCL info values are the same as the
00071        OpenCL ones used in Boost.Compute to just cast the enum class
00072        to the int value */
00073     return p.get_info<std::string>(static_cast<cl_platform_info>(param));
00074   }
```

8.3.2.13.2.4 has_extension()

```
bool cl::sycl::detail::opencl_platform::has_extension (
            const string_class & extension ) const  [inline], [override], [virtual]
```

Specify whether a specific extension is supported on the platform.

Implements cl::sycl::detail::platform.

Definition at line 78 of file opencl_platform.hpp.

```
00078                                                                   {
00079     return p.supports_extension(extension);
00080   }
```

8.3.2.13.2.5 instance()

```
static std::shared_ptr<opencl_platform> cl::sycl::detail::opencl_platform::instance (
            const boost::compute::platform & p )  [inline], [static]
```

Definition at line 85 of file opencl_platform.hpp.

References get_devices(), and cl::sycl::detail::cache< Key, Value >::get_or_register().

Referenced by cl::sycl::platform::platform().

```
00085                                                   {
00086     return cache.get_or_register(p.id(),
00087                                  [&] { return new opencl_platform {
      p }; });
00088   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.2.13.2.6 is_host()

```
bool cl::sycl::detail::opencl_platform::is_host ( ) const  [inline], [override], [virtual]
```

Return false since an OpenCL platform is not the SYCL host platform.

Implements cl::sycl::detail::platform.

Definition at line 63 of file opencl_platform.hpp.

```
00063                                   {
00064     return false;
00065   }
```

8.3.2.13.3 Member Data Documentation

8.3.2.13.3.1 cache

```
detail::cache<cl_platform_id, detail::opencl_platform> cl::sycl::detail::opencl_platform↩
::cache  [static], [private]
```

A cache to always return the same live platform for a given OpenCL platform.

C++11 guaranties the static construction is thread-safe

Definition at line 46 of file opencl_platform.hpp.

Referenced by ~opencl_platform().

8.3.2.13.3.2 p

```
boost::compute::platform cl::sycl::detail::opencl_platform::p  [private]
```

Use the Boost Compute abstraction of the OpenCL platform.

Definition at line 39 of file opencl_platform.hpp.

Referenced by get_boost_compute().

8.3.2.14 class cl::sycl::detail::platform

An abstract class representing various models of SYCL platforms.

Definition at line 28 of file platform.hpp.

Inheritance diagram for cl::sycl::detail::platform:

Public Member Functions

- virtual cl_platform_id get () const =0

  Return the cl_platform_id of the underlying OpenCL platform.
- virtual const boost::compute::platform & get_boost_compute () const =0

  Return the underlying Boost.Compute platform, if any.
- virtual bool is_host () const =0

  Return true if the platform is a SYCL host platform.
- virtual string_class get_info_string (info::platform param) const =0

  Query the platform for OpenCL string info::platform info.
- virtual bool has_extension (const string_class &extension) const =0

  Specify whether a specific extension is supported on the platform.
- virtual vector_class< device > get_devices (info::device_type device_type) const =0

  Get all the available devices for this platform.
- virtual ~platform ()

8.3.2.14.1 Constructor & Destructor Documentation

8.3.2.14.1.1 ~platform()

```
virtual cl::sycl::detail::platform::~platform ( )  [inline], [virtual]
```

Definition at line 65 of file platform.hpp.

```
00065 {}
```

8.3.2.14.2 Member Function Documentation

8.3.2.14.2.1 get()

```
virtual cl_platform_id cl::sycl::detail::platform::get ( ) const  [pure virtual]
```

Return the cl_platform_id of the underlying OpenCL platform.

Implemented in cl::sycl::detail::opencl_platform, and cl::sycl::detail::host_platform.

8.3.2.14.2.2 get_boost_compute()

```
virtual const boost::compute::platform& cl::sycl::detail::platform::get_boost_compute ( )
const  [pure virtual]
```

Return the underlying Boost.Compute platform, if any.

Implemented in cl::sycl::detail::opencl_platform, and cl::sycl::detail::host_platform.

8.3.2.14.2.3 get_devices()

```
virtual vector_class<device> cl::sycl::detail::platform::get_devices (
            info::device_type device_type ) const  [pure virtual]
```

Get all the available devices for this platform.

Parameters

in	*device_type*	is the device type to filter the selection or `info::device_type::all` by default to return all the devices

Returns

the device list

Implemented in cl::sycl::detail::host_platform, and cl::sycl::detail::opencl_platform.

8.3.2.14.2.4 get_info_string()

```
virtual string_class cl::sycl::detail::platform::get_info_string (
            info::platform param ) const  [pure virtual]
```

Query the platform for OpenCL string info::platform info.

Implemented in cl::sycl::detail::host_platform, and cl::sycl::detail::opencl_platform.

8.3.2.14.2.5 has_extension()

```
virtual bool cl::sycl::detail::platform::has_extension (
            const string_class & extension ) const  [pure virtual]
```

Specify whether a specific extension is supported on the platform.

Implemented in cl::sycl::detail::host_platform, and cl::sycl::detail::opencl_platform.

8.3.2.14.2.6 is_host()

```
virtual bool cl::sycl::detail::platform::is_host ( ) const  [pure virtual]
```

Return true if the platform is a SYCL host platform.

Implemented in cl::sycl::detail::host_platform, and cl::sycl::detail::opencl_platform.

8.3.2.15 class cl::sycl::platform

Abstract the OpenCL platform.

Todo triSYCL Implementation

Definition at line 42 of file platform.hpp.

Inheritance diagram for cl::sycl::platform:

Collaboration diagram for cl::sycl::platform:

Public Member Functions

- platform ()

  Default constructor for platform which is the host platform.
- platform (cl_platform_id platform_id)

  Construct a platform class instance using cl_platform_id of the OpenCL device.
- platform (const boost::compute::platform &p)

  Construct a platform class instance using a boost::compute::platform.
- platform (const device_selector &dev_selector)

  Construct a platform object from the device selected by a device selector of the user's choice.
- cl_platform_id get () const

  Returns the cl_platform_id of the underlying OpenCL platform.
- const boost::compute::platform get_boost_compute () const

  Return the underlying Boost.Compute platform if it is an OpenCL platform.
- template<typename ReturnT >
  ReturnT get_info (info::platform param) const

  Get the OpenCL information about the requested parameter.
- template<info::platform Param>
  info::param_traits< info::platform, Param >::return_type get_info () const

  Get the OpenCL information about the requested template parameter.
- bool has_extension (const string_class &extension) const

  Test if an extension is available on the platform.
- bool is_host () const

  Test if this platform is a host platform.
- vector_class< device > get_devices (info::device_type device_type=info::device_type::all) const

  Get all the available devices for this platform.

Static Public Member Functions

- static vector_class< platform > get_platforms ()

  Get the list of all the platforms available to the application.

Private Attributes

- friend shared_ptr_implementation

Additional Inherited Members

8.3.2.15.1 Constructor & Destructor Documentation

8.3.2.15.1.1 platform() [1/4]

```
cl::sycl::platform::platform ( ) [inline]
```

Default constructor for platform which is the host platform.

Returns errors via the SYCL exception class.

Definition at line 59 of file platform.hpp.

References cl::sycl::detail::singleton< host_platform >::instance().

Referenced by cl::sycl::detail::opencl_device::get_platform().

```
00059              :
00060     shared_ptr_implementation {
      detail::host_platform::instance() } {}
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.2.15.1.2 platform() [2/4]

```
cl::sycl::platform::platform (
            cl_platform_id platform_id )  [inline]
```

Construct a platform class instance using cl_platform_id of the OpenCL device.

Return synchronous errors via the SYCL exception class.

Retain a reference to the OpenCL platform.

Definition at line 71 of file platform.hpp.

```
00072     : platform { boost::compute::platform { platform_id } } {}
```

8.3.2.15.1.3 platform() [3/4]

```
cl::sycl::platform::platform (
            const boost::compute::platform & p )  [inline]
```

Construct a platform class instance using a boost::compute::platform.

This is a triSYCL extension for boost::compute interoperation.

Return synchronous errors via the SYCL exception class.

Definition at line 81 of file platform.hpp.

References cl::sycl::detail::opencl_platform::instance().

```
00082     : shared_ptr_implementation {
      detail::opencl_platform::instance(p) } {}
```

Here is the call graph for this function:

8.3.2.15.1.4 platform() [4/4]

```
cl::sycl::platform::platform (
            const device_selector & dev_selector )  [inline], [explicit]
```

Construct a platform object from the device selected by a device selector of the user's choice.

Returns errors via the SYCL exception class.

Definition at line 91 of file platform.hpp.

References cl::sycl::detail::unimplemented().

```
00091                                                   {
00092     detail::unimplemented();
00093   }
```

Here is the call graph for this function:

8.3.2.15.2 Member Function Documentation

8.3.2.15.2.1 get()

```
cl_platform_id cl::sycl::platform::get ( ) const  [inline]
```

Returns the cl_platform_id of the underlying OpenCL platform.

If the platform is not a valid OpenCL platform, for example if it is the SYCL host, an exception is thrown

Todo Define a SYCL exception for this

Definition at line 104 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00104                                {
00105     return implementation->get();
00106   }
```

8.3.2.15.2.2 get_boost_compute()

```
const boost::compute::platform cl::sycl::platform::get_boost_compute ( ) const  [inline]
```

Return the underlying Boost.Compute platform if it is an OpenCL platform.

This is a triSYCL extension

Definition at line 114 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00114                                                    {
00115     return implementation->get_boost_compute();
00116   }
```

8.3.2.15.2.3 get_devices()

```
vector_class<device> cl::sycl::platform::get_devices (
            info::device_type device_type = info::device_type::all ) const  [inline]
```

Get all the available devices for this platform.

Parameters

in	*device_type*	is the device type to filter the selection or `info::device_type::all` by default to return all the devices

Returns

the device list

Definition at line 180 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00180                                                                    {
00181     return implementation->get_devices(device_type);
00182   }
```

8.3.2.15.2.4 get_info() [1/2]

```
template<typename ReturnT >
ReturnT cl::sycl::platform::get_info (
            info::platform param ) const  [inline]
```

Get the OpenCL information about the requested parameter.

Todo Add to the specification

Definition at line 140 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00140                                                {
00141     // Only strings are needed here
00142     return implementation->get_info_string(param);
00143   }
```

8.3.2.15.2.5 get_info() [2/2]

```
template<info::platform Param>
info::param_traits<info::platform, Param>::return_type cl::sycl::platform::get_info ( ) const
[inline]
```

Get the OpenCL information about the requested template parameter.

Definition at line 149 of file platform.hpp.

```
00149                    {
00150     /* Forward to the implementation without using template parameter
00151        but with a parameter instead, since it is incompatible with
00152        virtual function and because fortunately only strings are
00153        needed here */
00154     return get_info<typename info::param_traits<info::platform,
00155                                                 Param>::return_type>(Param);
00156   }
```

8.3.2.15.2.6 get_platforms()

```
static vector_class<platform> cl::sycl::platform::get_platforms ( )  [inline], [static]
```

Get the list of all the platforms available to the application.

Definition at line 121 of file platform.hpp.

```
00121                                                {
00122     // Start with the default platform
00123     vector_class<platform> platforms { {} };
00124
00125 #ifdef TRISYCL_OPENCL
00126     // Then add all the OpenCL platforms
00127     for (const auto &d : boost::compute::system::platforms())
00128       platforms.emplace_back(d);
00129 #endif
00130
00131     return platforms;
00132   }
```

8.3.2.15.2.7 has_extension()

```
bool cl::sycl::platform::has_extension (
            const string_class & extension ) const  [inline]
```

Test if an extension is available on the platform.

Definition at line 160 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00160                                                    {
00161     return implementation->has_extension(extension);
00162   }
```

8.3.2.15.2.8 is_host()

```
bool cl::sycl::platform::is_host ( ) const  [inline]
```

Test if this platform is a host platform.

Definition at line 166 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< platform, detail::platform >::implementation.

```
00166                        {
00167     return implementation->is_host();
00168   }
```

8.3.2.15.3 Member Data Documentation

8.3.2.15.3.1 shared_ptr_implementation

```
friend cl::sycl::platform::shared_ptr_implementation  [private]
```

Definition at line 48 of file platform.hpp.

8.3.2.16 class cl::sycl::queue

SYCL queue, similar to the OpenCL queue concept.

Todo The implementation is quite minimal for now. :-)

Todo All the queue methods should return a queue& instead of void to it is possible to chain opoerations

Definition at line 80 of file queue.hpp.

Inheritance diagram for cl::sycl::queue:

Collaboration diagram for cl::sycl::queue:

Public Member Functions

- queue ()

  Default constructor for platform which is the host platform.
- queue (async_handler asyncHandler)

  This constructor creates a SYCL queue from an OpenCL queue.
- queue (const device_selector &deviceSelector, async_handler asyncHandler=nullptr)

  Creates a queue for the device provided by the device selector.
- queue (const device &d, async_handler asyncHandler=nullptr)

  A queue is created for a SYCL device.
- queue (const context &syclContext, const device_selector &deviceSelector, async_handler asyncHandler=nullptr)

  This constructor chooses a device based on the provided device_selector, which needs to be in the given context.

- queue (const context &syclContext, const device &syclDevice, async_handler asyncHandler=nullptr)

  Creates a command queue using clCreateCommandQueue from a context and a device.
- queue (const context &syclContext, const device &syclDevice, info::queue_profiling profilingFlag, async_handler asyncHandler=nullptr)

  Creates a command queue using clCreateCommandQueue from a context and a device.
- queue (const cl_command_queue &q, async_handler ah=nullptr)

  This constructor creates a SYCL queue from an OpenCL queue.
- queue (const boost::compute::command_queue &q, async_handler ah=nullptr)

  Construct a queue instance using a boost::compute::command_queue.
- cl_command_queue get () const

  Return the underlying OpenCL command queue after doing a retain.
- boost::compute::command_queue get_boost_compute () const

  Return the underlying Boost.Compute command queue if it is an OpenCL queue.
- context get_context () const

  Return the SYCL queue's context.
- device get_device () const

  Return the SYCL device the queue is associated with.
- bool is_host () const

  Return whether the queue is executing on a SYCL host device.
- void wait ()

  Performs a blocking wait for the completion all enqueued tasks in the queue.
- void wait_and_throw ()

  Perform a blocking wait for the completion all enqueued tasks in the queue.
- void throw_asynchronous ()

  Checks to see if any asynchronous errors have been produced by the queue and if so reports them by passing them to the async_handler passed to the queue on construction.
- template<info::queue param>
  info::param_traits< info::queue, param >::type get_info () const

  Queries the platform for cl_command_queue info.
- template<typename Handler_Functor >
  handler_event submit (Handler_Functor cgf)

  Submit a command group functor to the queue, in order to be scheduled for execution on the device.
- handler_event submit (std::function< void(handler &)> cgf, queue &secondaryQueue)

  Submit a command group functor to the queue, in order to be scheduled for execution on the device.

Private Types

- using implementation_t = typename queue::shared_ptr_implementation

Private Attributes

- friend implementation_t

Additional Inherited Members

8.3.2.16.1 Member Typedef Documentation

8.3.2.16.1.1 implementation_t

```
using cl::sycl::queue::implementation_t = typename queue::shared_ptr_implementation  [private]
```

Definition at line 87 of file queue.hpp.

8.3.2.16.2 Constructor & Destructor Documentation

8.3.2.16.2.1 queue() [1/9]

```
cl::sycl::queue::queue ( )  [inline]
```

Default constructor for platform which is the host platform.

Returns errors via the SYCL exception class.

Todo Check with the specification if it is the host queue or the one related to the default device selector.

Definition at line 107 of file queue.hpp.

```
00107 : implementation_t { new detail::host_queue } {}
```

8.3.2.16.2.2 queue() [2/9]

```
cl::sycl::queue::queue (
            async_handler asyncHandler )  [inline], [explicit]
```

This constructor creates a SYCL queue from an OpenCL queue.

At construction it does a retain on the queue memory object.

Retain a reference to the cl_command_queue object. Caller should release the passed cl_command_queue object when it is no longer needed.

Return synchronous errors regarding the creation of the queue and report asynchronous errors via the async_↩handler callback function in conjunction with the synchronization and throw methods.

Note that the default case asyncHandler = nullptr is handled by the default constructor.

Definition at line 126 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00126                                                 : queue { } {
00127     detail::unimplemented();
00128   }
```

Here is the call graph for this function:

8.3.2.16.2.3 queue() [3/9]

```
cl::sycl::queue::queue (
            const device_selector & deviceSelector,
            async_handler asyncHandler = nullptr )  [inline]
```

Creates a queue for the device provided by the device selector.

If no device is selected, an error is reported.

Return synchronous errors regarding the creation of the queue and report asynchronous errors via the `async_handler` callback function if and only if there is an `async_handler` provided.

Definition at line 139 of file queue.hpp.

```
00142     : queue { device { deviceSelector }, asyncHandler} { }
```

8.3.2.16.2.4 queue() [4/9]

```
cl::sycl::queue::queue (
            const device & d,
            async_handler asyncHandler = nullptr )  [inline]
```

A queue is created for a SYCL device.

Return asynchronous errors via the `async_handler` callback function.

Definition at line 150 of file queue.hpp.

References cl::sycl::detail::opencl_queue::instance(), and cl::sycl::device::is_host().

```
00151                                                 : implementation_t {
00152 #ifdef TRISYCL_OPENCL
00153     d.is_host()
00154       ? std::shared_ptr<detail::queue>{ new detail::host_queue }
00155       : detail::opencl_queue::instance(d)
00156 #else
00157     new detail::host_queue
00158 #endif
00159
00160   } { }
```

Here is the call graph for this function:

8.3.2.16.2.5 queue() [5/9]

```
cl::sycl::queue::queue (
            const context & syclContext,
            const device_selector & deviceSelector,
            async_handler asyncHandler = nullptr )  [inline]
```

This constructor chooses a device based on the provided device_selector, which needs to be in the given context.

If no device is selected, an error is reported.

Return synchronous errors regarding the creation of the queue.

If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 174 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00176                                                  : queue { } {
00177     detail::unimplemented();
00178   }
```

Here is the call graph for this function:

8.3.2.16.2.6 queue() [6/9]

```
cl::sycl::queue::queue (
            const context & syclContext,
            const device & syclDevice,
            async_handler asyncHandler = nullptr )  [inline]
```

Creates a command queue using clCreateCommandQueue from a context and a device.

Return synchronous errors regarding the creation of the queue.

If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 190 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00192                                                   : queue { } {
00193     detail::unimplemented();
00194   }
```

Here is the call graph for this function:

8.3.2.16.2.7 queue() [7/9]

```
cl::sycl::queue::queue (
            const context & syclContext,
            const device & syclDevice,
            info::queue_profiling profilingFlag,
            async_handler asyncHandler = nullptr )  [inline]
```

Creates a command queue using clCreateCommandQueue from a context and a device.

It enables profiling on the queue if the profilingFlag is set to true.

Return synchronous errors regarding the creation of the queue. If and only if there is an asyncHandler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 208 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00211                                                   : queue { } {
00212     detail::unimplemented();
00213   }
```

Here is the call graph for this function:

8.3.2.16.2.8 queue() [8/9]

```
cl::sycl::queue::queue (
            const cl_command_queue & q,
            async_handler ah = nullptr )  [inline]
```

This constructor creates a SYCL queue from an OpenCL queue.

At construction it does a retain on the queue memory object.

Return synchronous errors regarding the creation of the queue. If and only if there is an async_handler provided, it reports asynchronous errors via the async_handler callback function in conjunction with the synchronization and throw methods.

Definition at line 226 of file queue.hpp.

```
00227     : queue { boost::compute::command_queue { q }, ah } {}
```

8.3.2.16.2.9 queue() [9/9]

```
cl::sycl::queue::queue (
            const boost::compute::command_queue & q,
            async_handler ah = nullptr )  [inline]
```

Construct a queue instance using a boost::compute::command_queue.

This is a triSYCL extension for boost::compute interoperation.

Return synchronous errors via the SYCL exception class.

Todo Deal with handler

Definition at line 238 of file queue.hpp.

References cl::sycl::detail::opencl_queue::instance().

```
00239     : implementation_t { detail::opencl_queue::instance(q) }
      {}
```

Here is the call graph for this function:

8.3.2.16.3 Member Function Documentation

8.3.2.16.3.1 get()

```
cl_command_queue cl::sycl::queue::get ( ) const  [inline]
```

Return the underlying OpenCL command queue after doing a retain.

This memory object is expected to be released by the developer.

Retain a reference to the returned cl_command_queue object.

Caller should release it when finished.

If the queue is a SYCL host queue then an exception is thrown.

Definition at line 254 of file queue.hpp.

```
00254                                {
00255     return implementation->get();
00256   }
```

8.3.2.16.3.2 get_boost_compute()

```
boost::compute::command_queue cl::sycl::queue::get_boost_compute ( ) const  [inline]
```

Return the underlying Boost.Compute command queue if it is an OpenCL queue.

This is a triSYCL extension

Definition at line 264 of file queue.hpp.

```
00264                                                    {
00265     return implementation->get_boost_compute();
00266   }
```

8.3.2.16.3.3 get_context()

```
context cl::sycl::queue::get_context ( ) const  [inline]
```

Return the SYCL queue's context.

Report errors using SYCL exception classes.

Definition at line 274 of file queue.hpp.

```
00274                                {
00275     return implementation->get_context();
00276   }
```

8.3.2.16.3.4 get_device()

```
device cl::sycl::queue::get_device ( ) const  [inline]
```

Return the SYCL device the queue is associated with.

Report errors using SYCL exception classes.

Definition at line 283 of file queue.hpp.

```
00283                                {
00284     return implementation->get_device();
00285   }
```

8.3.2.16.3.5 get_info()

```
template<info::queue param>
info::param_traits<info::queue, param>::type cl::sycl::queue::get_info ( ) const  [inline]
```

Queries the platform for cl_command_queue info.

Definition at line 335 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00335                                                                    {
00336     detail::unimplemented();
00337     return {};
00338   }
```

Here is the call graph for this function:

8.3.2.16.3.6 is_host()

```
bool cl::sycl::queue::is_host ( ) const  [inline]
```

Return whether the queue is executing on a SYCL host device.

Definition at line 289 of file queue.hpp.

```
00289                           {
00290     return implementation->is_host();
00291   }
```

8.3.2.16.3.7 submit() [1/2]

```
template<typename Handler_Functor >
handler_event cl::sycl::queue::submit (
            Handler_Functor cgf )  [inline]
```

Submit a command group functor to the queue, in order to be scheduled for execution on the device.

Use an explicit functor parameter taking a handler& so we can use "auto" in submit() lambda parameter.

Todo Add in the spec an implicit conversion of handler_event to queue& so it is possible to chain operations on the queue

Definition at line 351 of file queue.hpp.

```
00351                                                {
00352     handler command_group_handler { implementation };
00353     cgf(command_group_handler);
00354     return {};
00355   }
```

8.3.2.16.3.8 submit() [2/2]

```
handler_event cl::sycl::queue::submit (
            std::function< void(handler &)> cgf,
            queue & secondaryQueue )  [inline]
```

Submit a command group functor to the queue, in order to be scheduled for execution on the device.

On kernel error, this command group functor, then it is scheduled for execution on the secondary queue.

Return a command group functor event, which is corresponds to the queue the command group functor is being enqueued on.

Definition at line 367 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00367                                                                                    {
00368     detail::unimplemented();
00369     // Since it is not implemented, always submit on the main queue
00370     return submit(cgf);
00371   }
```

Here is the call graph for this function:

8.3.2.16.3.9 throw_asynchronous()

```
void cl::sycl::queue::throw_asynchronous ( ) [inline]
```

Checks to see if any asynchronous errors have been produced by the queue and if so reports them by passing them to the async_handler passed to the queue on construction.

If no async_handler was provided then asynchronous exceptions will be lost.

Definition at line 328 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00328                                {
00329     detail::unimplemented();
00330   }
```

Here is the call graph for this function:

8.3.2.16.3.10 wait()

```
void cl::sycl::queue::wait ( ) [inline]
```

Performs a blocking wait for the completion all enqueued tasks in the queue.

Synchronous errors will be reported through SYCL exceptions.

Definition at line 299 of file queue.hpp.

```
00299                {
00300     implementation->wait_for_kernel_execution();
00301   }
```

8.3.2.16.3.11 wait_and_throw()

```
void cl::sycl::queue::wait_and_throw ( ) [inline]
```

Perform a blocking wait for the completion all enqueued tasks in the queue.

Synchronous errors will be reported via SYCL exceptions.

Asynchronous errors will be passed to the async_handler passed to the queue on construction.

If no async_handler was provided then asynchronous exceptions will be lost.

Definition at line 314 of file queue.hpp.

References cl::sycl::detail::unimplemented().

```
00314                         {
00315     // \todo Implement the throw part of wait_and_throw
00316     wait();
00317     detail::unimplemented();
00318   }
```

Here is the call graph for this function:

8.3.2.16.4 Member Data Documentation

8.3.2.16.4.1 implementation_t

```
friend cl::sycl::queue::implementation_t [private]
```

Definition at line 93 of file queue.hpp.

8.3.3 Typedef Documentation

8.3.3.1 cpu_selector

```
using cl::sycl::cpu_selector = typedef device_typename_selector<info::device_type::cpu>
```

```
#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>
```

Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.

If no OpenCL CPU device is found the selector fails.

Definition at line 133 of file device_selector_tail.hpp.

8.3.3.2 default_selector

```
using cl::sycl::default_selector = typedef device_typename_selector<info::device_type::automatic>
```

```
#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>
```

Devices selected by heuristics of the system.

If no OpenCL device is found then it defaults to the SYCL host device.

To influence the default device selection, use the Boost.Compute environment variables:

- BOOST_COMPUTE_DEFAULT_DEVICE
- BOOST_COMPUTE_DEFAULT_DEVICE_TYPE
- BOOST_COMPUTE_DEFAULT_PLATFORM
- BOOST_COMPUTE_DEFAULT_VENDOR

Definition at line 115 of file device_selector_tail.hpp.

8.3.3.3 device_exec_capabilities

```
using cl::sycl::info::device_exec_capabilities = typedef unsigned int
```

```
#include <include/CL/sycl/info/device.hpp>
```

Definition at line 183 of file device.hpp.

8.3.3.4 device_fp_config

```
using cl::sycl::info::device_fp_config = typedef unsigned int
```

```
#include <include/CL/sycl/info/device.hpp>
```

Definition at line 182 of file device.hpp.

8.3.3.5 device_queue_properties

```
using cl::sycl::info::device_queue_properties = typedef unsigned int
```

```
#include <include/CL/sycl/info/device.hpp>
```

Definition at line 184 of file device.hpp.

8.3.3.6 gpu_selector

```
using cl::sycl::gpu_selector = typedef device_typename_selector<info::device_type::gpu>
```

```
#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>
```

Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.

If no OpenCL GPU device is found the selector fails.

Select the best GPU, if any.

Definition at line 125 of file device_selector_tail.hpp.

8.3.3.7 host_selector

```
using cl::sycl::host_selector = typedef device_typename_selector<info::device_type::host>
```

```
#include <include/CL/sycl/device_selector/detail/device_selector_tail.hpp>
```

Selects the SYCL host CPU device that does not require an OpenCL runtime.

Definition at line 139 of file device_selector_tail.hpp.

8.3.4 Enumeration Type Documentation

8.3.4.1 context

```
enum cl::sycl::info::context : int [strong]
```

```
#include <include/CL/sycl/info/context.hpp>
```

Context information descriptors.

Todo Should be unsigned int to be consistent with others?

Enumerator

reference_count	
platform	
devices	

Definition at line 26 of file context.hpp.

```
00026                    : int {
00027   reference_count,
00028   platform,
00029   devices
00030 };
```

8.3.4.2 device

```
enum cl::sycl::info::device :  int  [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Device information descriptors.

From specs/latex/headers/deviceInfo.h in the specification

Todo Should be unsigned int?

Enumerator

device_type	
vendor_id	
max_compute_units	
max_work_item_dimensions	
max_work_item_sizes	
max_work_group_size	
preferred_vector_width_char	
preferred_vector_width_short	
preferred_vector_width_int	
preferred_vector_width_long_long	
preferred_vector_width_float	
preferred_vector_width_double	
preferred_vector_width_half	
native_vector_witdth_char	
native_vector_witdth_short	
native_vector_witdth_int	
native_vector_witdth_long_long	
native_vector_witdth_float	
native_vector_witdth_double	
native_vector_witdth_half	

Enumerator

max_clock_frequency	
address_bits	
max_mem_alloc_size	
image_support	
max_read_image_args	
max_write_image_args	
image2d_max_height	
image2d_max_width	
image3d_max_height	
image3d_max_widht	
image3d_mas_depth	
image_max_buffer_size	
image_max_array_size	
max_samplers	
max_parameter_size	
mem_base_addr_align	
single_fp_config	
double_fp_config	
global_mem_cache_type	
global_mem_cache_line_size	
global_mem_cache_size	
global_mem_size	
max_constant_buffer_size	
max_constant_args	
local_mem_type	
local_mem_size	
error_correction_support	
host_unified_memory	
profiling_timer_resolution	
endian_little	
is_available	
is_compiler_available	
is_linker_available	
execution_capabilities	
queue_properties	
built_in_kernels	
platform	
name	
vendor	
driver_version	
profile	
device_version	
opencl_version	
extensions	
printf_buffer_size	
preferred_interop_user_sync	
parent_device	
partition_max_sub_devices	

Enumerator

partition_properties	
partition_affinity_domain	
partition_type	
reference_count	

Definition at line 52 of file device.hpp.

```
00052                    : int {
00053   device_type,
00054   vendor_id,
00055   max_compute_units,
00056   max_work_item_dimensions,
00057   max_work_item_sizes,
00058   max_work_group_size,
00059   preferred_vector_width_char,
00060   preferred_vector_width_short,
00061   preferred_vector_width_int,
00062   preferred_vector_width_long_long,
00063   preferred_vector_width_float,
00064   preferred_vector_width_double,
00065   preferred_vector_width_half,
00066   native_vector_witdth_char,
00067   native_vector_witdth_short,
00068   native_vector_witdth_int,
00069   native_vector_witdth_long_long,
00070   native_vector_witdth_float,
00071   native_vector_witdth_double,
00072   native_vector_witdth_half,
00073   max_clock_frequency,
00074   address_bits,
00075   max_mem_alloc_size,
00076   image_support,
00077   max_read_image_args,
00078   max_write_image_args,
00079   image2d_max_height,
00080   image2d_max_width,
00081   image3d_max_height,
00082   image3d_max_widht,
00083   image3d_mas_depth,
00084   image_max_buffer_size,
00085   image_max_array_size,
00086   max_samplers,
00087   max_parameter_size,
00088   mem_base_addr_align,
00089   single_fp_config,
00090   double_fp_config,
00091   global_mem_cache_type,
00092   global_mem_cache_line_size,
00093   global_mem_cache_size,
00094   global_mem_size,
00095   max_constant_buffer_size,
00096   max_constant_args,
00097   local_mem_type,
00098   local_mem_size,
00099   error_correction_support,
00100   host_unified_memory,
00101   profiling_timer_resolution,
00102   endian_little,
00103   is_available,
00104   is_compiler_available,
00105   is_linker_available,
00106   execution_capabilities,
00107   queue_properties,
00108   built_in_kernels,
00109   platform,
00110   name,
00111   vendor,
00112   driver_version,
00113   profile,
00114   device_version,
00115   opencl_version,
00116   extensions,
00117   printf_buffer_size,
00118   preferred_interop_user_sync,
00119   parent_device,
00120   partition_max_sub_devices,
00121   partition_properties,
```

```
00122   partition_affinity_domain,
00123   partition_type,
00124   reference_count
00125 };
```

8.3.4.3 device_affinity_domain

`enum cl::sycl::info::device_affinity_domain : int [strong]`

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

unsupported	
numa	
L4_cache	
L3_cache	
L2_cache	
next_partitionable	

Definition at line 135 of file device.hpp.

```
00135                                   : int {
00136   unsupported,
00137   numa,
00138   L4_cache,
00139   L3_cache,
00140   L2_cache,
00141   next_partitionable
00142 };
```

8.3.4.4 device_execution_capabilities

`enum cl::sycl::info::device_execution_capabilities : unsigned int [strong]`

`#include <include/CL/sycl/info/device.hpp>`

Enumerator

exec_kernel	
exec_native_kernel	

Definition at line 176 of file device.hpp.

```
00176                                           : unsigned int {
00177   exec_kernel,
00178   exec_native_kernel
00179 };
```

8.3.4.5 device_partition_property

```
enum cl::sycl::info::device_partition_property : int [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Enumerator

unsupported	
partition_equally	
partition_by_counts	
partition_by_affinity_domain	
partition_affinity_domain_next_partitionable	

Definition at line 127 of file device.hpp.

```
00127                                       : int {
00128   unsupported,
00129   partition_equally,
00130   partition_by_counts,
00131   partition_by_affinity_domain,
00132   partition_affinity_domain_next_partitionable
00133 };
```

8.3.4.6 device_partition_type

```
enum cl::sycl::info::device_partition_type : int [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Enumerator

no_partition	
numa	
L4_cache	
L3_cache	
L2_cache	
L1_cache	

Definition at line 144 of file device.hpp.

```
00144                                     : int {
00145   no_partition,
00146   numa,
00147   L4_cache,
00148   L3_cache,
00149   L2_cache,
00150   L1_cache
00151 };
```

8.3.4.7 device_type

```
enum cl::sycl::info::device_type : unsigned int [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Type of devices.

To be used either to define a device type or to select more broadly a kind of device

Todo To be moved in the specification from platform to device

Todo Add opencl to the specification

Todo there is no accelerator_selector and custom_accelerator

Enumerator

cpu	
gpu	
accelerator	
custom	
automatic	
host	
opencl	
all	

Definition at line 34 of file device.hpp.

```
00034                          : unsigned int {
00035   cpu,
00036   gpu,
00037   accelerator,
00038   custom,
00039   automatic,
00040   host,
00041   opencl,
00042   all
00043 };
```

8.3.4.8 fp_config

```
enum cl::sycl::info::fp_config : int [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Enumerator

denorm	
inf_nan	

Enumerator

round_to_nearest	
round_to_zero	
round_to_inf	
fma	
correctly_rounded_divide_sqrt	
soft_float	

Definition at line 159 of file device.hpp.

```
00159                       : int {
00160   denorm,
00161   inf_nan,
00162   round_to_nearest,
00163   round_to_zero,
00164   round_to_inf,
00165   fma,
00166   correctly_rounded_divide_sqrt,
00167   soft_float
00168 };
```

8.3.4.9 global_mem_cache_type

```
enum cl::sycl::info::global_mem_cache_type :  int  [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Enumerator

none	
read_only	
write_only	

Definition at line 170 of file device.hpp.

```
00170                                   : int {
00171   none,
00172   read_only,
00173   write_only
00174 };
```

8.3.4.10 local_mem_type

```
enum cl::sycl::info::local_mem_type :  int  [strong]
```

```
#include <include/CL/sycl/info/device.hpp>
```

Enumerator

none	
local	
global	

Definition at line 153 of file device.hpp.

```
00153                                   : int {
00154   none,
00155   local,
00156   global
00157 };
```

8.3.4.11 platform

```
enum cl::sycl::info::platform :  unsigned int  [strong]
```

```
#include <include/CL/sycl/info/platform.hpp>
```

Platform information descriptors.

A SYCL platform can be queried for all of the following information using the get_info function.

In this implementation, the values are mapped to OpenCL values to avoid further remapping later when OpenCL is used

Enumerator

TRISYCL_SKIP_OPENCL	Returns the profile name (as a string_class) supported by the implementation. Can be either FULL PROFILE or EMBEDDED PROFILE.
TRISYCL_SKIP_OPENCL	Returns the OpenCL software driver version string in the form major number.minor number (as a string_class)
TRISYCL_SKIP_OPENCL	Returns the name of the platform (as a string_class)
TRISYCL_SKIP_OPENCL	Returns the string provided by the platform vendor (as a string_class)
TRISYCL_SKIP_OPENCL	Returns a space-separated list of extension names supported by the platform (as a string_class)

Definition at line 31 of file platform.hpp.

```
00031                             : unsigned int {
00032   /** Returns the profile name (as a string_class) supported by the
00033       implementation.
00034
00035       Can be either FULL PROFILE or EMBEDDED PROFILE.
00036   */
00037   profile TRISYCL_SKIP_OPENCL(= CL_PLATFORM_PROFILE),
00038
00039   /** Returns the OpenCL software driver version string in the form major
00040       number.minor number (as a string_class)
00041   */
00042   version TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VERSION),
00043
00044   /** Returns the name of the platform (as a string_class)
00045   */
```

```
  name TRISYCL_SKIP_OPENCL(= CL_PLATFORM_NAME),

  /** Returns the string provided by the platform vendor (as a string_class)
  */
  vendor TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VENDOR),

  /** Returns a space-separated list of extension names supported by the
      platform (as a string_class)
  */
  extensions TRISYCL_SKIP_OPENCL(= CL_PLATFORM_EXTENSIONS),

#if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
  /** Returns the resolution of the host timer in nanoseconds as used by
      clGetDeviceAndHostTimer
  */
  host_timer_resolution
    TRISYCL_SKIP_OPENCL(= CL_PLATFORM_HOST_TIMER_RESOLUTION)
#endif
};
```

8.3.5 Function Documentation

8.3.5.1 device::get_info< info::device::device_type >()

```
template<>
auto cl::sycl::device::get_info< info::device::device_type > ( ) const  [inline]
```

#include <include/CL/sycl/device.hpp>

Definition at line 270 of file device.hpp.

References cl::sycl::info::cpu.

```
{
  return info::device_type::cpu;
}
```

8.3.5.2 device::get_info< info::device::local_mem_size >()

```
template<>
auto cl::sycl::device::get_info< info::device::local_mem_size > ( ) const  [inline]
```

#include <include/CL/sycl/device.hpp>

Definition at line 275 of file device.hpp.

```
{
  return size_t { 32000 };
}
```

8.3.5.3 device::get_info< info::device::max_compute_units >()

```
template<>
auto cl::sycl::device::get_info< info::device::max_compute_units > ( ) const  [inline]
```

```
#include <include/CL/sycl/device.hpp>
```

Definition at line 265 of file device.hpp.

```
00265                                                                    {
00266   return size_t { 8 };
00267 }
```

8.3.5.4 device::get_info< info::device::max_mem_alloc_size >()

```
template<>
auto cl::sycl::device::get_info< info::device::max_mem_alloc_size > ( ) const  [inline]
```

```
#include <include/CL/sycl/device.hpp>
```

Definition at line 280 of file device.hpp.

```
00280                                                                    {
00281   return size_t { 32000 };
00282 }
```

8.3.5.5 device::get_info< info::device::max_work_group_size >()

```
template<>
auto cl::sycl::device::get_info< info::device::max_work_group_size > ( ) const  [inline]
```

```
#include <include/CL/sycl/device.hpp>
```

Definition at line 260 of file device.hpp.

```
00260                                                                    {
00261   return size_t { 8 };
00262 }
```

8.3.5.6 device::get_info< info::device::name >()

```
template<>
auto cl::sycl::device::get_info< info::device::name > ( ) const  [inline]
```

#include <include/CL/sycl/device.hpp>

Definition at line 290 of file device.hpp.

```
00290                                                    {
00291   return string_class {};
00292 }
```

8.3.5.7 device::get_info< info::device::profile >()

```
template<>
auto cl::sycl::device::get_info< info::device::profile > ( ) const  [inline]
```

#include <include/CL/sycl/device.hpp>

Definition at line 295 of file device.hpp.

```
00295                                                     {
00296   return string_class { "FULL_PROFILE" };
00297 }
```

8.3.5.8 device::get_info< info::device::vendor >()

```
template<>
auto cl::sycl::device::get_info< info::device::vendor > ( ) const  [inline]
```

#include <include/CL/sycl/device.hpp>

Definition at line 285 of file device.hpp.

```
00285                                                    {
00286   return string_class {};
00287 }
```

8.3.5.9 get_devices() [1/3]

```
vector_class< cl::sycl::device > cl::sycl::detail::opencl_platform::get_devices (
            info::device_type device_type ) const  [inline], [override], [virtual]
```

#include <include/CL/sycl/platform/detail/opencl_platform.hpp>

Get all the available devices for this OpenCL platform.

Returns a vector class containing all SYCL devices associated with this OpenCL platform.

Parameters

in	*device_type*	is the device type to filter the selection or `info::device_type::all` by default to return all the devices

Returns

the device list

Implements cl::sycl::detail::platform.

Definition at line 32 of file opencl_platform_tail.hpp.

References cl::sycl::detail::opencl_platform::get_boost_compute().

Referenced by cl::sycl::detail::opencl_platform::instance().

```
00032                                                                    {
00033   vector_class<cl::sycl::device> devices;
00034     device_type_selector ds { device_type };
00035   // Add the desired OpenCL devices
00036   for (const auto &d : get_boost_compute().devices()) {
00037     // Get the SYCL device from the Boost Compute device
00038     cl::sycl::device sycl_dev { d };
00039     /* Return the devices with the good criterion according to the selector.
00040        By calling devices on the \c boost::compute::platform we know that
00041        we iterate only over the device belonging to the current platform,
00042     */
00043     if (ds(sycl_dev) > 0)
00044       devices.push_back(sycl_dev);
00045   }
00046
00047   return devices;
00048 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.3.5.10 get_devices() [2/3]

```
vector_class< cl::sycl::device > cl::sycl::detail::host_platform::get_devices (
            info::device_type device_type ) const  [inline], [override], [virtual]
```

```
#include <include/CL/sycl/platform/detail/host_platform.hpp>
```

Get all the available devices for the host platform.

Get all the available devices for this platform.

Parameters

in	*device_type*	is the device type to filter the selection or `info::device_type::all` by default to return all the devices

Returns

the device list

If `get_devices` is called with the host platform and the right device type, returns the host_device.

Implements cl::sycl::detail::platform.

Definition at line 31 of file host_platform_tail.hpp.

Referenced by cl::sycl::detail::host_platform::has_extension().

```
00031                                                                    {
00032   /** If \c get_devices is called with the host platform
00033       and the right device type, returns the host_device.
00034   */
00035   if (device_type_selector { device_type }(cl::sycl::device {}) > 0)
00036     // Return 1 default device, i.e. the host device
00037     return { {} };
00038   else
00039     // No matching device
00040     return {};
00041 }
```

Here is the caller graph for this function:

8.3.5.11 get_devices() [3/3]

```
vector_class< device > cl::sycl::device::get_devices (
            info::device_type device_type = info::device_type::all )  [static]
```

#include <include/CL/sycl/device.hpp>

Return a list of all available devices.

Return synchronous errors via SYCL exception classes.

Definition at line 26 of file device_tail.hpp.

Referenced by cl::sycl::device::device(), and cl::sycl::device::get_platform().

```
00026                                                   {
00027   // Start with the default device
00028   vector_class<device> devices = { {} };
00029
00030 #ifdef TRISYCL_OPENCL
00031   // Then add all the OpenCL devices
00032   for (const auto &d : boost::compute::system::devices())
00033     devices.emplace_back(d);
00034 #endif
00035
00036   // The selected devices
00037   vector_class<device> sd;
00038   device_type_selector s { device_type };
00039
00040   // Return the devices with the good criterion according to the selector
00041   std::copy_if(devices.begin(), devices.end(), std::back_inserter(sd),
00042                [&](const device &e) { return s(e) >= 0; });
00043   return sd;
00044 }
```

Here is the caller graph for this function:

8.3.6 Variable Documentation

8.3.6.1 TRISYCL_WEAK_ATTRIB_SUFFIX

```
TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_command_queue, detail::opencl_queue > opencl_↩
queue::cache cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX
```

#include <include/CL/sycl/context/detail/opencl_context.hpp>

Definition at line 142 of file opencl_context.hpp.

Referenced by cl::sycl::detail::__attribute__(), cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_RANG↩E(), cl::sycl::detail::opencl_device::~opencl_device(), cl::sycl::detail::opencl_platform::~opencl_platform(), and cl↩::sycl::detail::opencl_queue::~opencl_queue().

8.4 Helpers to do array and tuple conversion

Classes

- struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > cl::sycl::detail::tuple_to_array_iterate (Tuple t, std↩::index_sequence< Is... >)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- static auto cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand (Tuple t)
- template<typename Value , size_t... Is>
  static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::fill_tuple (Value e, std::index_sequence< Is... >)

  Construct a tuple from a value.
- static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::expand (Tuple t)

  We expand the 1-element tuple by replicating into a tuple with the size of the vector.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::expand (Tuple t)

  Create the array data of V from a tuple of initializer.

8.4.1 Detailed Description

8.4.2 Class Documentation

8.4.2.1 struct cl::sycl::detail::expand_to_vector

template<typename V, typename Tuple, bool expansion = false>
struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization.

Definition at line 65 of file array_tuple_helpers.hpp.

Static Public Member Functions

- static auto expand (Tuple t)

8.4.2.2 struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

template<typename V, typename Tuple>
struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

Specialization in the case we ask for expansion.

Definition at line 77 of file array_tuple_helpers.hpp.

Static Public Member Functions

- template<typename Value , size_t... Is>
  static auto fill_tuple (Value e, std::index_sequence< Is... >)

  Construct a tuple from a value.
- static auto expand (Tuple t)

  We expand the 1-element tuple by replicating into a tuple with the size of the vector.

8.4.3 Function Documentation

8.4.3.1 expand() [1/3]

```
template<typename V , typename Tuple , bool expansion = false>
static auto cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand (
            Tuple t )  [inline], [static]
```

#include <include/CL/sycl/detail/array_tuple_helpers.hpp>

Definition at line 70 of file array_tuple_helpers.hpp.

Referenced by cl::sycl::detail::expand().

```
00070 { return t; }
```

Here is the caller graph for this function:

8.4.3.2 expand() [2/3]

```
template<typename V , typename Tuple >
static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::expand (
            Tuple t )  [inline], [static]
```

#include <include/CL/sycl/detail/array_tuple_helpers.hpp>

We expand the 1-element tuple by replicating into a tuple with the size of the vector.

Definition at line 109 of file array_tuple_helpers.hpp.

```
00109                                {
00110     return fill_tuple(std::get<0>(t),
00111                       std::make_index_sequence<V::dimension>{});
00112   }
```

8.4.3.3 expand() [3/3]

```
template<typename V , typename Tuple >
auto cl::sycl::detail::expand (
            Tuple t )
```

#include <include/CL/sycl/detail/array_tuple_helpers.hpp>

Create the array data of V from a tuple of initializer.

If there is only 1 initializer, this is a scalar initialization of a vector and the value is expanded to all the vector elements first.

Definition at line 123 of file array_tuple_helpers.hpp.

References cl::sycl::detail::expand_to_vector< V, Tuple, expansion >::expand().

```
00123                      {
00124   return tuple_to_array<V>(expand_to_vector<V,
00125                            decltype(t),
00126                            /* Only ask the expansion to all vector
00127                               element if there only a scalar
00128                               initializer */
00129                            std::tuple_size<Tuple>::value == 1>{}.expand(t));
00130 }
```

Here is the call graph for this function:

8.4.3.4 fill_tuple()

```
template<typename V , typename Tuple >
template<typename Value , size_t...  Is>
static auto cl::sycl::detail::expand_to_vector< V, Tuple, true >::fill_tuple (
            Value e,
            std::index_sequence< Is...  >  )  [inline], [static]
```

#include <include/CL/sycl/detail/array_tuple_helpers.hpp>

Construct a tuple from a value.

Parameters

value	is used to initialize each tuple element
size	is the number of elements of the tuple to be generated

The trick is to get the std::index_sequence<> that represent 0, 1,..., dimension-1 as a variadic template pack Is that we can iterate on, in this function.

Definition at line 93 of file array_tuple_helpers.hpp.

```
00093                                                                {
00094     /* The effect is like a static for-loop with Is counting from 0 to
00095        dimension-1 and thus replicating the pattern to have
00096        make_tuple( (0, e), (1, e), ... (n - 1, e) )
00097
00098        Since the "," operator is just here to throw away the Is value
00099        (which is needed for the pack expansion...), at the end this is
00100        equivalent to:
00101        make_tuple( e, e, ..., e )
00102     */
00103     return std::make_tuple(((void)Is, e)...);
00104   }
```

8.4.3.5 tuple_to_array()

```
template<typename V , typename Tuple >
auto cl::sycl::detail::tuple_to_array (
            Tuple t )
```

#include <include/CL/sycl/detail/array_tuple_helpers.hpp>

Construct an array from initializer elements provided as a tuple.

Definition at line 53 of file array_tuple_helpers.hpp.

```
00053                                {
00054   /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
00055      so that tuple_to_array_iterate can statically iterate on it */
00056   return tuple_to_array_iterate<V>(t,
00057                                    std::make_index_sequence<std::tuple_size<Tuple>::value>{});
00058 }
```

8.4.3.6 tuple_to_array_iterate()

```
template<typename V , typename Tuple , size_t...  Is>
std::array<typename V::element_type, V::dimension> cl::sycl::detail::tuple_to_array_iterate (
            Tuple t,
            std::index_sequence< Is...  >  )
```

#include <include/CL/sycl/detail/array_tuple_helpers.hpp>

Helper to construct an array from initializer elements provided as a tuple.

The trick is to get the std::index_sequence<> that represent 0, 1,..., dimension-1 as a variadic template pack Is that we can iterate on, in this function.

Definition at line 37 of file array_tuple_helpers.hpp.

```
00037                                                                 {
00038   /* The effect is like a static for-loop with Is counting from 0 to
00039      dimension-1 and thus constructing a uniform initialization { }
00040      construction from each tuple element:
00041      { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }
00042
00043      The static cast is here to avoid the warning when there is a loss
00044      of precision, for example when initializing an int from a float.
00045   */
00046   return { { static_cast<typename V::element_type>(std::get<Is>(t))...} };
00047 }
```

8.5 Some helpers for the implementation

Classes

- struct cl::sycl::detail::container_element_aspect< T >

  A mix-in to add some container element aspects. More...
- struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if Dimensions = 1. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Macros

- #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)

  Helper macro to declare a vector operation with the given side-effect operator.
- #define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(op)

Functions

- template<typename Range , typename Id >
  size_t constexpr cl::sycl::detail::linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.
- void cl::sycl::detail::unimplemented ()

  Display an "unimplemented" message.

8.5.1 Detailed Description

8.5.2 Class Documentation

8.5.2.1 struct cl::sycl::detail::container_element_aspect

template<typename T>
struct cl::sycl::detail::container_element_aspect< T >

A mix-in to add some container element aspects.

Definition at line 23 of file container_element_aspect.hpp.

Public Types

- using value_type = T
- using pointer = value_type *
- using const_pointer = const value_type *
- using reference = value_type &
- using const_reference = const value_type &

8.5.2.1.1 Member Typedef Documentation

8.5.2.1.1.1 const_pointer

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::const_pointer = const value_type*
```

Definition at line 27 of file container_element_aspect.hpp.

8.5.2.1.1.2 const_reference

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::const_reference = const value_type&
```

Definition at line 29 of file container_element_aspect.hpp.

8.5.2.1.1.3 pointer

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::pointer = value_type*
```

Definition at line 26 of file container_element_aspect.hpp.

8.5.2.1.1.4 reference

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::reference = value_type&
```

Definition at line 28 of file container_element_aspect.hpp.

8.5.2.1.1.5 value_type

```
template<typename T>
using cl::sycl::detail::container_element_aspect< T >::value_type = T
```

Definition at line 25 of file container_element_aspect.hpp.

8.5.2.2 struct cl::sycl::detail::small_array

template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor = false>
struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

Define a multi-dimensional index, used for example to locate a work item or a buffer element.

Unfortunately, even if std::array is an aggregate class allowing native list initialization, it is no longer an aggregate if we derive from an aggregate. Thus we have to redeclare the constructors.

Parameters

BasicType	is the type element, such as int
Dims	is the dimension number, typically between 1 and 3
FinalType	is the final type, such as range<> or id<>, so that boost::operator can return the right type
EnableArgsConstructor	adds a constructors from Dims variadic elements when true. It is false by default.

std::array<> provides the collection concept, with .size(), == and != too.

Definition at line 74 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:

Collaboration diagram for cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >:

Public Types

- using element_type = BasicType

Public Member Functions

- template<typename SourceType >
  small_array (const SourceType src[Dims])

  A constructor from another array.
- BasicType & x ()

  An accessor to the first variable of a small array.
- BasicType & y ()

  An accessor to the second variable of a small array.
- BasicType & z ()

  An accessor to the third variable of a small array.
- template<typename SourceBasicType , typename SourceFinalType , bool SourceEnableArgsConstructor>
  small_array (const small_array< SourceBasicType, SourceFinalType, Dims, SourceEnableArgsConstructor > &src)

  A constructor from another small_array of the same size.
- template<typename... Types, bool Depend = true, typename = typename std::enable_if_t<EnableArgsConstructor && Depend>>
  small_array (const Types &... args)

  Initialize the array from a list of elements.
- template<typename SourceBasicType >
  small_array (const std::array< SourceBasicType, Dims > &src)

  Construct a small_array from a std::array.
- small_array ()=default

  Keep the synthesized constructors.
- auto get (std::size_t index) const

  Return the element of the array.
- operator FinalType ()

  Add + like operations on the id<> and others.

Static Public Attributes

- static const auto dimensionality = Dims
- static const size_t dimension = Dims

8.5.2.2.1 Member Typedef Documentation

8.5.2.2.1.1 element_type

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
using cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >↩
::element_type = BasicType
```

Definition at line 94 of file small_array.hpp.

8.5.2.2.2 Constructor & Destructor Documentation

8.5.2.2.2.1 small_array() [1/5]

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
template<typename SourceType >
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_↩
array (
            const SourceType src[Dims] )  [inline]
```

A constructor from another array.

Make it explicit to avoid spurious range<> constructions from int * for example

Definition at line 103 of file small_array.hpp.

```
00103                                         {
00104     // (*this)[0] is the first element of the underlying array
00105     std::copy_n(src, Dims, &(*this)[0]);
00106   }
```

8.5.2.2.2.2 small_array() [2/5]

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
template<typename SourceBasicType , typename SourceFinalType , bool SourceEnableArgsConstructor>
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_↩
array (
            const small_array< SourceBasicType, SourceFinalType, Dims, SourceEnableArgs↩
Constructor > & src )  [inline]
```

A constructor from another small_array of the same size.

Definition at line 137 of file small_array.hpp.

```
00140                                                 {
00141     std::copy_n(&src[0], Dims, &(*this)[0]);
00142   }
```

8.5.2.2.2.3 small_array() [3/5]

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
template<typename... Types, bool Depend = true, typename = typename std::enable_if_t<Enable↩
ArgsConstructor && Depend>>
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_↩
array (
            const Types &... args ) [inline]
```

Initialize the array from a list of elements.

Strangely, even when using the array constructors, the initialization of the aggregate is not available. So recreate an equivalent here.

Since there are inherited types that defines some constructors with some conflicts, make it optional here, according to EnableArgsConstructor template parameter.

Definition at line 160 of file small_array.hpp.

```
00161     : std::array<BasicType, Dims> {
00162     // Allow a loss of precision in initialization with the static_cast
00163     { static_cast<BasicType>(args)... }
00164   }
```

8.5.2.2.2.4 small_array() [4/5]

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
template<typename SourceBasicType >
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_↩
array (
            const std::array< SourceBasicType, Dims > & src ) [inline]
```

Construct a small_array from a std::array.

Definition at line 174 of file small_array.hpp.

```
00175   : std::array<BasicType, Dims>(src) {}
```

8.5.2.2.2.5 small_array() [5/5]

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::small_↩
array ( ) [default]
```

Keep the synthesized constructors.

Referenced by cl::sycl::detail::small_array< BasicType, FinalType, 3 >::small_array().

Here is the caller graph for this function:

8.5.2.2.3 Member Function Documentation

8.5.2.2.3.1 get()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
auto cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::get (
            std::size_t index ) const  [inline]
```

Return the element of the array.

Definition at line 185 of file small_array.hpp.

```
00185                                   {
00186     return (*this)[index];
00187   }
```

8.5.2.2.3.2 operator FinalType()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::operator
FinalType ( )  [inline]
```

Add + like operations on the id<> and others.

Add - like operations on the id<> and others Add ∗ like operations on the id<> and others Add / like operations on the id<> and others Add % like operations on the id<> and others Add << like operations on the id<> and others Add >> like operations on the id<> and others Add & like operations on the id<> and others Add ^ like operations on the id<> and others Add | like operations on the id<> and others Since the boost::operator work on the small_array, add an implicit conversion to produce the expected type

Definition at line 228 of file small_array.hpp.

```
00228                           {
00229     return *static_cast<FinalType *>(this);
00230   }
```

8.5.2.2.3.3 x()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
BasicType& cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::x ( ) [inline]
```

An accessor to the first variable of a small array.

Definition at line 111 of file small_array.hpp.

Referenced by cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123(), cl::sycl::detail↩ ::small_array_123< BasicType, FinalType, 2 >::small_array_123(), and cl::sycl::detail::small_array_123< Basic↩ Type, FinalType, 3 >::small_array_123().

```
00111                 {
00112     static_assert(Dims >= 1, "can't access to small_array[0] if Dims < 1");
00113     return (*this)[0];
00114   }
```

Here is the caller graph for this function:

8.5.2.2.3.4 y()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
BasicType& cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::y ( ) [inline]
```

An accessor to the second variable of a small array.

Definition at line 119 of file small_array.hpp.

Referenced by cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123(), and cl::sycl↩ ::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123().

```
00119                 {
00120     static_assert(Dims >= 2, "can't access to small_array[1] if Dims < 2");
00121     return (*this)[1];
00122   }
```

Here is the caller graph for this function:

8.5.2.2.3.5 z()

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
BasicType& cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::z ( )  [inline]
```

An accessor to the third variable of a small array.

Definition at line 127 of file small_array.hpp.

Referenced by cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123().

```
00127                 {
00128     static_assert(Dims >= 3, "can't access to small_array[2] if Dims < 3");
00129     return (*this)[2];
00130   }
```

Here is the caller graph for this function:

8.5.2.2.4 Member Data Documentation

8.5.2.2.4.1 dimension

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
const size_t cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::dimension = Dims  [static]
```

Definition at line 93 of file small_array.hpp.

8.5.2.2.4.2 dimensionality

```
template<typename BasicType, typename FinalType, std::size_t Dims, bool EnableArgsConstructor
= false>
const auto cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor
>::dimensionality = Dims  [static]
```

Todo add this Boost::multi_array or STL concept to the specification?

Definition at line 89 of file small_array.hpp.

8.5.2.3 struct cl::sycl::detail::small_array_123

template<typename BasicType, typename FinalType, std::size_t Dims>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

A small array of 1, 2 or 3 elements with the implicit constructors.

Definition at line 237 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >:

Additional Inherited Members

8.5.2.4 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if Dimensions = 1.

Definition at line 249 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >:

Public Member Functions

- small_array_123 (BasicType x)

  A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.
- small_array_123 ()=default

  Keep other constructors.
- operator BasicType () const

  Conversion so that an for example an id<1> can basically be used like an integer.

Additional Inherited Members

8.5.2.4.1 Constructor & Destructor Documentation

8.5.2.4.1.1 small_array_123() [1/2]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 (
            BasicType x )  [inline]
```

A 1-D constructor to have implicit conversion from from 1 integer and automatic inference of the dimensionality.

Definition at line 253 of file small_array.hpp.

References cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::x().

```
00253                                {
00254     (*this)[0] = x;
00255   }
```

Here is the call graph for this function:

8.5.2.4.1.2 small_array_123() [2/2]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::small_array_123 ( ) [default]
```

Keep other constructors.

8.5.2.4.2 Member Function Documentation

8.5.2.4.2.1 operator BasicType()

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >::operator BasicType ( ) const
[inline]
```

Conversion so that an for example an id<1> can basically be used like an integer.

Definition at line 265 of file small_array.hpp.

```
00265                                {
00266     return (*this)[0];
00267   }
```

8.5.2.5 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >

template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >

Definition at line 272 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >:

Public Member Functions

- small_array_123 (BasicType x, BasicType y)

  A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.
- small_array_123 (BasicType e)

  Broadcasting constructor initializing all the elements with the same value.
- small_array_123 ()=default

  Keep other constructors.

Additional Inherited Members

8.5.2.5.1 Constructor & Destructor Documentation

8.5.2.5.1.1 small_array_123() [1/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (
            BasicType x,
            BasicType y )  [inline]
```

A 2-D constructor to have implicit conversion from from 2 integers and automatic inference of the dimensionality.

Definition at line 276 of file small_array.hpp.

References cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::x(), and cl::sycl↩::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::y().

```
00276                                                   {
00277     (*this)[0] = x;
00278     (*this)[1] = y;
00279   }
```

Here is the call graph for this function:

8.5.2.5.1.2 small_array_123() [2/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 (
            BasicType e )  [inline], [explicit]
```

Broadcasting constructor initializing all the elements with the same value.

Todo Add to the specification of the range, id...

Definition at line 287 of file small_array.hpp.

```
00287 : small_array_123 { e, e } { }
```

8.5.2.5.1.3 small_array_123() [3/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >::small_array_123 ( ) [default]
```

Keep other constructors.

8.5.2.6 struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

template<typename BasicType, typename FinalType>
struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Definition at line 298 of file small_array.hpp.

Inheritance diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >:

Collaboration diagram for cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >:

Public Member Functions

- small_array_123 (BasicType x, BasicType y, BasicType z)

  A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.
- small_array_123 (BasicType e)

  Broadcasting constructor initializing all the elements with the same value.
- small_array_123 ()=default

  Keep other constructors.

Additional Inherited Members

8.5.2.6.1 Constructor & Destructor Documentation

8.5.2.6.1.1 small_array_123() [1/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (
            BasicType x,
            BasicType y,
            BasicType z )  [inline]
```

A 3-D constructor to have implicit conversion from from 3 integers and automatic inference of the dimensionality.

Definition at line 302 of file small_array.hpp.

References cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::x(), cl::sycl↩::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::y(), and cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >::z().

```
00302                                                   {
00303     (*this)[0] = x;
00304     (*this)[1] = y;
00305     (*this)[2] = z;
00306   }
```

Here is the call graph for this function:

8.5.2.6.1.2 small_array_123() [2/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 (
            BasicType e )  [inline], [explicit]
```

Broadcasting constructor initializing all the elements with the same value.

Todo Add to the specification of the range, id...

Definition at line 314 of file small_array.hpp.

```
00314 : small_array_123 { e, e, e } { }
```

8.5.2.6.1.3 small_array_123() [3/3]

```
template<typename BasicType , typename FinalType >
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >::small_array_123 ( )  [default]
```

Keep other constructors.

8.5.3 Macro Definition Documentation

8.5.3.1 TRISYCL_BOOST_OPERATOR_VECTOR_OP

```
#define TRISYCL_BOOST_OPERATOR_VECTOR_OP(
              op )
```

#include <include/CL/sycl/detail/small_array.hpp>

Value:

```
FinalType operator op(const FinalType &rhs) {          \
    for (std::size_t i = 0; i != Dims; ++i)              \
      (*this)[i] op rhs[i];                              \
    return *this;                                        \
  }
```

Helper macro to declare a vector operation with the given side-effect operator.

Definition at line 33 of file small_array.hpp.

Referenced by cl::sycl::detail::small_array< BasicType, FinalType, 3 >::get().

8.5.3.2 TRISYCL_LOGICAL_OPERATOR_VECTOR_OP

```
#define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(
              op )
```

#include <include/CL/sycl/detail/small_array.hpp>

Value:

```
FinalType operator op(const FinalType &rhs) {          \
    FinalType res;                                       \
    for (std::size_t i = 0; i != Dims; ++i)              \
      res[i] = (*this)[i] op rhs[i];                     \
    return res;                                          \
  }
```

Definition at line 41 of file small_array.hpp.

Referenced by cl::sycl::detail::small_array< BasicType, FinalType, 3 >::get().

8.5.4 Function Documentation

8.5.4.1 linear_id()

```
template<typename Range , typename Id >
size_t constexpr cl::sycl::detail::linear_id (
            Range range,
            Id id,
            Id offset = {} )  [inline]
```

#include <include/CL/sycl/detail/linear_id.hpp>

Compute a linearized array access used in the OpenCL 2 world.

Typically for the get_global_linear_id() and get_local_linear_id() functions.

Definition at line 28 of file linear_id.hpp.

References i, and offset.

Referenced by cl::sycl::nd_item< Dimensions >::get_global_linear_id(), cl::sycl::nd_item< Dimensions >::get_group_linear_id(), cl::sycl::group< Dimensions >::get_linear(), cl::sycl::item< Dimensions >::get_linear_id(), and cl::sycl::nd_item< Dimensions >::get_local_linear_id().

```
00028                                                                   {}) {
00029   auto dims = std::distance(std::begin(range), std::end(range));
00030
00031   size_t linear_id = 0;
00032   /* A good compiler should unroll this and do partial evaluation to
00033      remove the first multiplication by 0 of this Horner evaluation and
00034      remove the 0 offset evaluation */
00035     for (int i = dims - 1; i >= 0; --i)
00036       linear_id = linear_id*range[i] + id[i] - offset[i];
00037
00038     return linear_id;
00039   }
```

Here is the caller graph for this function:

8.5.4.2 unimplemented()

```
void cl::sycl::detail::unimplemented ( ) [inline]
```

```
#include <include/CL/sycl/detail/unimplemented.hpp>
```

Display an "unimplemented" message.

Can be changed to call assert(0) or whatever.

Definition at line 25 of file unimplemented.hpp.

Referenced by cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::accessor(), cl::sycl::nd_item< Dimensions >::barrier(), cl::sycl::buffer< T, Dimensions, Allocator >::buffer(), cl::sycl::context::context(), cl::sycl::kernel::get(), cl::sycl::detail::opencl_kernel::get_boost_compute(), cl::sycl::detail::opencl_context::get_devices(), cl::sycl::context::get_devices(), cl::sycl::context::get_info(), cl::sycl::queue::get_info(), cl::sycl::detail::host_context::get_platform(), cl::sycl::detail::opencl_device::get_platform(), cl::sycl::detail::host_device::get_platform(), cl::sycl::detail::opencl_context::get_platform(), cl::sycl::detail::host_platform::has_extension(), cl::sycl::detail::opencl_device::has_extension(), cl::sycl::detail::host_device::has_extension(), cl::sycl::detail::opencl_context::is_host(), cl::sycl::platform::platform(), cl::sycl::queue::queue(), cl::sycl::queue::submit(), cl::sycl::queue::throw_asynchronous(), and cl::sycl::queue::wait_and_throw().

```
                                {
#ifndef NDEBUG
  std::cerr << "Error: using a non implemented feature!!!" << std::endl
            << "Please contribute to the open source implementation. :-)"
            << std::endl;
#endif
}
```

Here is the caller graph for this function:

8.6 Debugging and tracing support

Classes

- struct cl::sycl::detail::debug< T >

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- struct cl::sycl::detail::display_vector< T >

  Class used to display a vector-like type of classes that inherit from it. More...

Functions

- template<typename KernelName , typename Functor >
  auto cl::sycl::detail::trace_kernel (Functor f)

  Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

8.6.1 Detailed Description

8.6.2 Class Documentation

8.6.2.1 struct cl::sycl::detail::debug

template<typename T>
struct cl::sycl::detail::debug< T >

Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it.

Parameters

T	is the real type name to be used in the debug output.

Definition at line 68 of file debug.hpp.

8.6.2.2 struct cl::sycl::detail::display_vector

template<typename T>
struct cl::sycl::detail::display_vector< T >

Class used to display a vector-like type of classes that inherit from it.

Parameters

T	is the real type name to be used in the debug output.

Calling the display() method dump the values on std::cout

Definition at line 160 of file debug.hpp.

Public Member Functions

- void display () const

  To debug and test.

8.6.2.2.1 Member Function Documentation

8.6.2.2.1.1 display()

```
template<typename T>
void cl::sycl::detail::display_vector< T >::display ( ) const  [inline]
```

To debug and test.

Definition at line 163 of file debug.hpp.

Referenced by cl::sycl::nd_range< Dimensions >::display().

```
00163                       {
00164 #ifdef TRISYCL_DEBUG
00165     std::cout << boost::typeindex::type_id<T>().pretty_name() << ":";
00166 #endif
00167     // Get a pointer to the real object
00168     for (auto e : *static_cast<const T *>(this))
00169       std::cout << " " << e;
00170     std::cout << std::endl;
00171   }
```

Here is the caller graph for this function:

8.6.3 Function Documentation

8.6.3.1 trace_kernel()

```
template<typename KernelName , typename Functor >
auto cl::sycl::detail::trace_kernel (
            Functor f )
```

#include <include/CL/sycl/detail/debug.hpp>

Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

Definition at line 130 of file debug.hpp.

References TRISYCL_INTERNAL_DUMP.

```
00130                                {
00131 #ifdef TRISYCL_TRACE_KERNEL
00132   // Inject tracing message around the kernel
00133   return [=] () mutable {
00134     /* Since the class KernelName may just be declared and not really
00135        defined, just use it through a class pointer to have
00136        typeid().name() not complaining */
00137     TRISYCL_INTERNAL_DUMP(
00138       "Kernel started "
00139       << boost::typeindex::type_id<KernelName *>().pretty_name());
00140     f();
00141     TRISYCL_INTERNAL_DUMP(
00142       "Kernel stopped "
00143       << boost::typeindex::type_id<KernelName *>().pretty_name());
00144   };
00145 #else
00146   // Identity by default
00147   return f;
00148 #endif
00149 }
```

8.7 Manage default configuration and types

Collaboration diagram for Manage default configuration and types:

Namespaces

- cl

  The vector type to be used as SYCL vector.

Macros

- #define CL_SYCL_LANGUAGE_VERSION 121

  This implements SYCL 1.2.1.
- #define TRISYCL_CL_LANGUAGE_VERSION 121

  This implements triSYCL 1.2.1.
- #define __SYCL_SINGLE_SOURCE__

  This source is compiled by a single source compiler.
- #define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
- #define SDX_KERNEL_PROGRAM_OWNING_BUG

  Define TRISYCL_OPENCL to add OpenCL.
- #define TRISYCL_SKIP_OPENCL(x) x

  A macro to keep some stuff in OpenCL mode.
- #define TRISYCL_DEVICE_ONLY(X)

8.7.1 Detailed Description

8.7.2 Macro Definition Documentation

8.7.2.1 __SYCL_SINGLE_SOURCE__

```
#define __SYCL_SINGLE_SOURCE__
```

```
#include <include/CL/sycl/detail/global_config.hpp>
```

This source is compiled by a single source compiler.

Definition at line 28 of file global_config.hpp.

8.7.2.2 CL_SYCL_LANGUAGE_VERSION

```
#define CL_SYCL_LANGUAGE_VERSION 121
```

#include <include/CL/sycl/detail/global_config.hpp>

This implements SYCL 1.2.1.

Definition at line 19 of file global_config.hpp.

8.7.2.3 SDX_KERNEL_PROGRAM_OWNING_BUG

```
#define SDX_KERNEL_PROGRAM_OWNING_BUG
```

#include <include/CL/sycl/detail/global_config.hpp>

Define TRISYCL_OPENCL to add OpenCL.

triSYCL can indeed work without OpenCL if only host support is needed.

Definition at line 44 of file global_config.hpp.

8.7.2.4 TRISYCL_CL_LANGUAGE_VERSION

```
#define TRISYCL_CL_LANGUAGE_VERSION 121
```

#include <include/CL/sycl/detail/global_config.hpp>

This implements triSYCL 1.2.1.

Definition at line 24 of file global_config.hpp.

8.7.2.5 TRISYCL_DEVICE_ONLY

```
#define TRISYCL_DEVICE_ONLY(
            X )
```

#include <include/CL/sycl/detail/global_config.hpp>

Definition at line 68 of file global_config.hpp.

8.7.2.6 TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE

```
#define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
```

#include <include/CL/sycl/detail/global_config.hpp>

Definition at line 33 of file global_config.hpp.

8.7.2.7 TRISYCL_SKIP_OPENCL

```
#define TRISYCL_SKIP_OPENCL(
            x ) x
```

#include <include/CL/sycl/detail/global_config.hpp>

A macro to keep some stuff in OpenCL mode.

Definition at line 55 of file global_config.hpp.

8.8 Device-side runtime implementation

Namespaces

- cl::sycl::drt

  The device-side runtime.
- cl::sycl::drt::code

  The binary code of the kernels.

Classes

- class cl::sycl::drt::accessor< Accessor >

  SYCL accessor seen from a device perspective. More...

Functions

- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::serialize_arg (detail::task &task, std::size_t index, void *arg, std::size_t arg_size)

  Set an argument value of the kernel.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::serialize_accessor_arg (detail::task &task, std::size_t index, void *arg, std::size_t arg_size)

  Set an accessor argument of the kernel.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::set_kernel (detail::task &task, const char *kernel_name, const char *kernel_short_name)

  Set the kernel for a later invocation.

Variables

- TRISYCL_WEAK_ATTRIB_PREFIX std::map< std::string, std::vector< unsigned char > > TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::kernel_IR

  The code of the kernels arranged by name.

8.8.1 Detailed Description

8.8.2 Class Documentation

8.8.2.1 class cl::sycl::drt::accessor

template<typename Accessor>
class cl::sycl::drt::accessor< Accessor >

SYCL accessor seen from a device perspective.

Definition at line 53 of file device_runtime.hpp.

Inheritance diagram for cl::sycl::drt::accessor< Accessor >:

Collaboration diagram for cl::sycl::drt::accessor< Accessor >:

Public Member Functions

- accessor (Accessor &a)

  Capture the minimal accessor features.
- auto & operator[ ] (std::size_t index) const noexcept

  use the accessor with integers à la []

Private Attributes

- TRISYCL_GLOBAL_AS Accessor::value_type * buffer

  The pointer to the data.
- std::size_t size

  The size of the accessor.

Additional Inherited Members

8.8.2.1.1 Constructor & Destructor Documentation

8.8.2.1.1.1 accessor()

```
template<typename Accessor >
cl::sycl::drt::accessor< Accessor >::accessor (
            Accessor & a )  [inline]
```

Capture the minimal accessor features.

Definition at line 77 of file device_runtime.hpp.

References a, TRISYCL_DUMP_T, and TRISYCL_GLOBAL_AS.

```
00077                          :
00078     /* Initialize the buffer with actually a \c cl_mem since it will be
00079        overridden with the kernel argument by the serializer at
00080        run-time, but at least from the kernel point of view it has to
00081        look like an \c Accessor::value_type * for the kernel
00082        outlining */
00083     buffer {
00084       reinterpret_cast<TRISYCL_GLOBAL_AS typename Accessor::value_type *>
00085         (a.implementation->get_order())
00086         },
00087     size { a.get_count() }
00088   {
00089 #ifndef TRISYCL_DEVICE
00090     /* Register the buffer address to be updated with the final
00091        version before the kernel set arg */
00092     TRISYCL_DUMP_T("drt::accessor(Accessor &a) : &a = "
00093                    << (void *) &a << '\n'
00094                    << "\t&buffer = " << (void *) &buffer
00095                    << ", Order = " << a.implementation->get_order());
00096 #endif
00097   }
```

8.8.2.1.2 Member Function Documentation

8.8.2.1.2.1 operator[]()

```
template<typename Accessor >
auto& cl::sycl::drt::accessor< Accessor >::operator[] (
            std::size_t index ) const  [inline], [noexcept]
```

use the accessor with integers à la []

Definition at line 101 of file device_runtime.hpp.

References TRISYCL_WEAK_ATTRIB_PREFIX, and TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00101                                                   {
00102     return buffer[index];
00103   }
```

8.8.2.1.3 Member Data Documentation

8.8.2.1.3.1 buffer

```
template<typename Accessor >
TRISYCL_GLOBAL_AS Accessor::value_type* cl::sycl::drt::accessor< Accessor >::buffer  [private]
```

The pointer to the data.

For now none of the following works yet for various reasons:

```
 cl::sycl::global_ptr<typename Accessor::value_type>
      buffer;
 cl::sycl::global<typename Accessor::value_type *>
      buffer;
__global typename Accessor::pointer buffer;
```

So use a more trivial approach

Definition at line 67 of file device_runtime.hpp.

8.8.2.1.3.2 size

```
template<typename Accessor >
std::size_t cl::sycl::drt::accessor< Accessor >::size  [private]
```

The size of the accessor.

The compiler should optimize it out if it is unused

Definition at line 72 of file device_runtime.hpp.

8.8.3 Function Documentation

8.8.3.1 serialize_accessor_arg()

```
TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::serialize_accessor↩
_arg (
            detail::task & task,
            std::size_t index,
            void * arg,
            std::size_t arg_size )
```

```
#include <include/CL/sycl/device_runtime.hpp>
```

Set an accessor argument of the kernel.

Parameters

in	*task*	is the implementation detail of a triSYCL kernel
in	*index*	is the order of the argument to set (0 is the first one)
in	*arg*	points to the argument value
in	*arg_size*	is the size of the argument

Definition at line 140 of file device_runtime.hpp.

References cl::sycl::detail::task::get_compute_buffer(), cl::sycl::detail::task::set_arg(), and TRISYCL_DUMP_T.

```
00143                                                {
00144   TRISYCL_DUMP_T("serialize_accessor_arg index = " << index
00145                  << ", size = " << arg_size << ", arg = " << arg);
00146 #ifdef TRISYCL_OPENCL
00147   /* The address of the argument has been actually initialized by the
00148      accessor constructor to the accessor order to relate an argument
00149      to its accessor */
00150   task.set_arg(index,
00151                task.get_compute_buffer(reinterpret_cast<std::size_t>(arg)));
00152 #endif
00153 }
```

Here is the call graph for this function:

8.8.3.2 serialize_arg()

```
TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::serialize_arg (
            detail::task & task,
            std::size_t index,
            void * arg,
            std::size_t arg_size )
```

```
#include <include/CL/sycl/device_runtime.hpp>
```

Set an argument value of the kernel.

Parameters

in	*task*	is the implementation detail of a triSYCL kernel
in	*index*	is the order of the argument to set (0 is the first one)
in	*arg*	points to the argument value
in	*arg_size*	is the size of the argument

Definition at line 119 of file device_runtime.hpp.

References cl::sycl::detail::task::set_arg(), TRISYCL_DUMP_T, TRISYCL_WEAK_ATTRIB_PREFIX, and TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00122                                   {
00123   TRISYCL_DUMP_T("serialize_arg index = " << index
00124                  << ", size = " << arg_size << ", arg = " << arg);
00125   task.set_arg(index, arg_size, arg);
00126 }
```

Here is the call graph for this function:

8.8.3.3 set_kernel()

```
TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::set_kernel (
            detail::task & task,
            const char * kernel_name,
            const char * kernel_short_name )
```

```
#include <include/CL/sycl/device_runtime.hpp>
```

Set the kernel for a later invocation.

Parameters

in	*task*	is the implementation detail of a triSYCL kernel
in	*kernel_name*	is set by the device compiler to identify the kernel to call

Definition at line 201 of file device_runtime.hpp.

References cl::sycl::detail::task::get_queue(), cl::sycl::drt::code::program::p, cl::sycl::detail::task::set_kernel(), and TRISYCL_DUMP_T.

```
00203                                           {
00204   TRISYCL_DUMP_T("set_kernel setting up " << kernel_name
00205                  << "\n\taka " << kernel_short_name);
00206   // \todo Add kernel caching per device
00207   // auto binary = kernel_IR.find(kernel_name);
00208   // if (binary == kernel_IR.end()) {
00209 #ifdef TRISYCL_OPENCL
00210   auto context = task.get_queue()->get_boost_compute().get_context();
```

```
  // Construct an OpenCL program from the precompiled kernel file
  auto
#ifdef SDX_KERNEL_PROGRAM_OWNING_BUG
    /* There is a bug in Xilinx OpenCL runtime where the cl_kernel
       does not keep alive its own cl_program */
    static
#endif
    program = boost::compute::program::create_with_binary
    (code::program::p->binary,
     code::program::p->binary_size,
     context);
  TRISYCL_DUMP_T("...on device with name "
                 << task.get_queue()->get_boost_compute().get_device().name());
  // Build the OpenCL program
  program.build();

  // Build a SYCL kernel from the OpenCL kernel
  cl::sycl::kernel k { boost::compute::kernel { program, kernel_short_name } };

  task.set_kernel(k.implementation);
#endif
}
```

Here is the call graph for this function:

8.8.4 Variable Documentation

8.8.4.1 kernel_IR

```
TRISYCL_WEAK_ATTRIB_PREFIX std::map<std::string, std::vector<unsigned char> > TRISYCL_WEAK_↩
ATTRIB_SUFFIX cl::sycl::drt::kernel_IR
```

```
#include <include/CL/sycl/device_runtime.hpp>
```

The code of the kernels arranged by name.

Todo use std::byte when moving to C++17

Definition at line 190 of file device_runtime.hpp.

8.9 Error handling

Namespaces

- cl::sycl::trisycl

Classes

- struct cl::sycl::error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...
- struct cl::sycl::exception_list

  Exception list to store several exceptions. More...
- class cl::sycl::exception

  Encapsulate a SYCL error information. More...
- class cl::sycl::cl_exception

  Returns the OpenCL error code encapsulated in the exception. More...
- struct cl::sycl::async_exception

  An error stored in an exception_list for asynchronous errors. More...
- class cl::sycl::runtime_error
- class cl::sycl::kernel_error

  Error that occurred before or while enqueuing the SYCL kernel. More...
- class cl::sycl::accessor_error

  Error regarding the cl::sycl::accessor objects defined. More...
- class cl::sycl::nd_range_error

  Error regarding the cl::sycl::nd_range specified for the SYCL kernel. More...
- class cl::sycl::event_error

  Error regarding associated cl::sycl::event objects. More...
- class cl::sycl::invalid_parameter_error

  Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda. More...
- class cl::sycl::device_error

  The SYCL device will trigger this exception on error. More...
- class cl::sycl::compile_program_error

  Error while compiling the SYCL kernel to a SYCL device. More...
- class cl::sycl::link_program_error

  Error while linking the SYCL kernel to a SYCL device. More...
- class cl::sycl::invalid_object_error

  Error regarding any memory objects being used inside the kernel. More...
- class cl::sycl::memory_allocation_error

  Error on memory allocation on the SYCL device for a SYCL kernel. More...
- class cl::sycl::pipe_error

  A failing pipe error will trigger this exception on error. More...
- class cl::sycl::platform_error

  The SYCL platform will trigger this exception on error. More...
- class cl::sycl::profiling_error

  The SYCL runtime will trigger this error if there is an error when profiling info is enabled. More...
- class cl::sycl::feature_not_supported

  Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on. More...
- class cl::sycl::non_cl_error

  Exception for an OpenCL operation requested in a non OpenCL area. More...

Typedefs

- using cl::sycl::exception_ptr = std::exception_ptr

  A shared pointer to an exception as in C++ specification.
- using cl::sycl::async_handler = function_class< void, exception_list >

8.9.1 Detailed Description

8.9.2 Class Documentation

8.9.2.1 struct cl::sycl::error_handler

User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler.

Definition at line 32 of file error_handler.hpp.

Inheritance diagram for cl::sycl::error_handler:

Collaboration diagram for cl::sycl::error_handler:

Public Member Functions

- virtual void report_error (exception &error)=0

  The method to define to be called in the case of an error.
- virtual ~error_handler ()=0

Static Public Attributes

- static trisycl::default_error_handler default_handler

  Add a default_handler to be used by default.

8.9.2.1.1 Constructor & Destructor Documentation

8.9.2.1.1.1 ~error_handler()

```
virtual cl::sycl::error_handler::~error_handler ( )  [pure virtual]
```

8.9.2.1.2 Member Function Documentation

8.9.2.1.2.1 report_error()

```
virtual void cl::sycl::error_handler::report_error (
            exception & error )  [pure virtual]
```

The method to define to be called in the case of an error.

Todo Add "virtual void" to the specification

Implemented in cl::sycl::trisycl::default_error_handler.

8.9.2.1.3 Member Data Documentation

8.9.2.1.3.1 default_handler

```
trisycl::default_error_handler cl::sycl::error_handler::default_handler  [static]
```

Add a default_handler to be used by default.

Todo add this concept to the specification?

Definition at line 43 of file error_handler.hpp.

8.9.2.2 struct cl::sycl::exception_list

Exception list to store several exceptions.

Todo Do we need to define it in SYCL or can we rely on plain C++17 one?

Definition at line 33 of file exception.hpp.

Inheritance diagram for cl::sycl::exception_list:

Collaboration diagram for cl::sycl::exception_list:

8.9.2.3 class cl::sycl::exception

Encapsulate a SYCL error information.

Definition at line 41 of file exception.hpp.

Inheritance diagram for cl::sycl::exception:

Collaboration diagram for cl::sycl::exception:

Public Member Functions

- exception (const string_class &message)

  Construct an exception with a message for internal use.
- string_class what () const

  Returns a descriptive string for the error, if available.

Private Attributes

- string_class message

  The error message to return.

8.9.2.3.1 Constructor & Destructor Documentation

8.9.2.3.1.1 exception()

```
cl::sycl::exception::exception (
            const string_class & message )  [inline]
```

Construct an exception with a message for internal use.

Definition at line 49 of file exception.hpp.

```
00049 : message { message } {}
```

8.9.2.3.2 Member Function Documentation

8.9.2.3.2.1 what()

```
string_class cl::sycl::exception::what ( ) const  [inline]
```

Returns a descriptive string for the error, if available.

Definition at line 52 of file exception.hpp.

```
00052                                {
00053     return message;
00054   }
```

8.9.2.3.3 Member Data Documentation

8.9.2.3.3.1 message

```
string_class cl::sycl::exception::message  [private]
```

The error message to return.

Definition at line 44 of file exception.hpp.

8.9.2.4 class cl::sycl::cl_exception

Returns the OpenCL error code encapsulated in the exception.

Definition at line 69 of file exception.hpp.

Inheritance diagram for cl::sycl::cl_exception:

Collaboration diagram for cl::sycl::cl_exception:

Public Member Functions

- cl_exception (const string_class &message, cl_int cl_code)

  Construct an exception with a message and OpenCL error code for internal use.
- cl_int get_cl_code () const

Private Attributes

- cl_int cl_code

  The OpenCL error code to return.

8.9.2.4.1 Constructor & Destructor Documentation

8.9.2.4.1.1 cl_exception()

```
cl::sycl::cl_exception::cl_exception (
            const string_class & message,
            cl_int cl_code )  [inline]
```

Construct an exception with a message and OpenCL error code for internal use.

Definition at line 80 of file exception.hpp.

```
00081     : exception { message }, cl_code { cl_code } {}
```

8.9.2.4.2 Member Function Documentation

8.9.2.4.2.1 get_cl_code()

```
cl_int cl::sycl::cl_exception::get_cl_code ( ) const  [inline]
```

Definition at line 84 of file exception.hpp.

```
00084                                {
00085     return cl_code;
00086   }
```

8.9.2.4.3 Member Data Documentation

8.9.2.4.3.1 cl_code

```
cl_int cl::sycl::cl_exception::cl_code  [private]
```

The OpenCL error code to return.

Definition at line 74 of file exception.hpp.

8.9.2.5 struct cl::sycl::async_exception

An error stored in an exception_list for asynchronous errors.

Definition at line 93 of file exception.hpp.

Inheritance diagram for cl::sycl::async_exception:

Collaboration diagram for cl::sycl::async_exception:

Additional Inherited Members

8.9.2.6 class cl::sycl::runtime_error

Definition at line 98 of file exception.hpp.

Inheritance diagram for cl::sycl::runtime_error:

Collaboration diagram for cl::sycl::runtime_error:

Additional Inherited Members

8.9.2.7 class cl::sycl::kernel_error

Error that occurred before or while enqueuing the SYCL kernel.

Definition at line 104 of file exception.hpp.

Inheritance diagram for cl::sycl::kernel_error:

Collaboration diagram for cl::sycl::kernel_error:

Additional Inherited Members

8.9.2.8 class cl::sycl::accessor_error

Error regarding the cl::sycl::accessor objects defined.

Definition at line 110 of file exception.hpp.

Inheritance diagram for cl::sycl::accessor_error:

Collaboration diagram for cl::sycl::accessor_error:

Additional Inherited Members

8.9.2.9 class cl::sycl::nd_range_error

Error regarding the cl::sycl::nd_range specified for the SYCL kernel.

Definition at line 116 of file exception.hpp.

Inheritance diagram for cl::sycl::nd_range_error:

Collaboration diagram for cl::sycl::nd_range_error:

Additional Inherited Members

8.9.2.10 class cl::sycl::event_error

Error regarding associated cl::sycl::event objects.

Definition at line 122 of file exception.hpp.

Inheritance diagram for cl::sycl::event_error:

Collaboration diagram for cl::sycl::event_error:

Additional Inherited Members

8.9.2.11 class cl::sycl::invalid_parameter_error

Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda.

Definition at line 130 of file exception.hpp.

Inheritance diagram for cl::sycl::invalid_parameter_error:

Collaboration diagram for cl::sycl::invalid_parameter_error:

Additional Inherited Members

8.9.2.12 class cl::sycl::device_error

The SYCL device will trigger this exception on error.

Definition at line 136 of file exception.hpp.

Inheritance diagram for cl::sycl::device_error:

Collaboration diagram for cl::sycl::device_error:

Additional Inherited Members

8.9.2.13 class cl::sycl::compile_program_error

Error while compiling the SYCL kernel to a SYCL device.

Definition at line 142 of file exception.hpp.

Inheritance diagram for cl::sycl::compile_program_error:

Collaboration diagram for cl::sycl::compile_program_error:

Additional Inherited Members

8.9.2.14 class cl::sycl::link_program_error

Error while linking the SYCL kernel to a SYCL device.

Definition at line 148 of file exception.hpp.

Inheritance diagram for cl::sycl::link_program_error:

Collaboration diagram for cl::sycl::link_program_error:

Additional Inherited Members

8.9.2.15 class cl::sycl::invalid_object_error

Error regarding any memory objects being used inside the kernel.

Definition at line 154 of file exception.hpp.

Inheritance diagram for cl::sycl::invalid_object_error:

Collaboration diagram for cl::sycl::invalid_object_error:

Additional Inherited Members

8.9.2.16 class cl::sycl::memory_allocation_error

Error on memory allocation on the SYCL device for a SYCL kernel.

Definition at line 160 of file exception.hpp.

Inheritance diagram for cl::sycl::memory_allocation_error:

Collaboration diagram for cl::sycl::memory_allocation_error:

Additional Inherited Members

8.9.2.17 class cl::sycl::pipe_error

A failing pipe error will trigger this exception on error.

Definition at line 166 of file exception.hpp.

Inheritance diagram for cl::sycl::pipe_error:

Collaboration diagram for cl::sycl::pipe_error:

Additional Inherited Members

8.9.2.18 class cl::sycl::platform_error

The SYCL platform will trigger this exception on error.

Definition at line 172 of file exception.hpp.

Inheritance diagram for cl::sycl::platform_error:

Collaboration diagram for cl::sycl::platform_error:

Additional Inherited Members

8.9.2.19 class cl::sycl::profiling_error

The SYCL runtime will trigger this error if there is an error when profiling info is enabled.

Definition at line 180 of file exception.hpp.

Inheritance diagram for cl::sycl::profiling_error:

Collaboration diagram for cl::sycl::profiling_error:

Additional Inherited Members

8.9.2.20 class cl::sycl::feature_not_supported

Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on.

Definition at line 189 of file exception.hpp.

Inheritance diagram for cl::sycl::feature_not_supported:

Collaboration diagram for cl::sycl::feature_not_supported:

Additional Inherited Members

8.9.2.21 class cl::sycl::non_cl_error

Exception for an OpenCL operation requested in a non OpenCL area.

Todo Add to the specification

Todo Clean implementation

Todo Exceptions are named error in C++

Definition at line 202 of file exception.hpp.

Inheritance diagram for cl::sycl::non_cl_error:

Collaboration diagram for cl::sycl::non_cl_error:

Additional Inherited Members

8.9.3 Typedef Documentation

8.9.3.1 async_handler

```
using cl::sycl::async_handler = typedef function_class<void, exception_list>
```

```
#include <include/CL/sycl/exception.hpp>
```

Definition at line 37 of file exception.hpp.

8.9.3.2 exception_ptr

```
using cl::sycl::exception_ptr = typedef std::exception_ptr
```

```
#include <include/CL/sycl/exception.hpp>
```

A shared pointer to an exception as in C++ specification.

Todo Do we need this instead of reusing directly the one from C++11?

Definition at line 26 of file exception.hpp.

8.10 Expressing parallelism through kernels

Collaboration diagram for Expressing parallelism through kernels:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Classes

- struct cl::sycl::group< Dimensions >

  A group index used in a parallel_for_workitem to specify a work_group. More...
- class cl::sycl::id< Dimensions >

  Define a multi-dimensional index, used for example to locate a work item. More...
- class cl::sycl::item< Dimensions >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- struct cl::sycl::nd_item< Dimensions >

  A SYCL nd_item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...
- struct cl::sycl::nd_range< Dimensions >

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

  A recursive multi-dimensional iterator that ends up calling f. More...
- struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...
- class cl::sycl::range< Dimensions >

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

Functions

- auto cl::sycl::make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_id (id< 2 > i)
- auto cl::sycl::make_id (id< 3 > i)
- template<typename... BasicType>
  auto cl::sycl::make_id (BasicType... Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)
- template<int Dimensions = 1, typename ParallelForFunctor , typename Id >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, Id)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, item< Dimensions >)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFunctor f)

  Implementation of parallel_for with a range<> and an offset.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- auto cl::sycl::make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_range (range< 2 > r)
- auto cl::sycl::make_range (range< 3 > r)
- template<typename... BasicType>
  auto cl::sycl::make_range (BasicType... Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

8.10.1 Detailed Description

8.10.2 Class Documentation

8.10.2.1 struct cl::sycl::group

template<int Dimensions>
struct cl::sycl::group< Dimensions >

A group index used in a parallel_for_workitem to specify a work_group.

Definition at line 24 of file group.hpp.

Collaboration diagram for cl::sycl::group< Dimensions >:

Public Member Functions

- group (const nd_range< Dimensions > &ndr)

  Create a group from an nd_range<> with a 0 id<>
- group (const id< Dimensions > &i, const nd_range< Dimensions > &ndr)

  Create a group from an id and a nd_range<>
- group ()=default

  To be able to copy and assign group, use default constructors too.
- id< Dimensions > get_id () const

  Return an id representing the index of the group within the nd_range for every dimension.
- size_t get_id (int dimension) const

  Return the index of the group in the given dimension.
- auto & operator[] (int dimension)

  Return the index of the group in the given dimension within the nd_range<>
- range< Dimensions > get_group_range () const

  Return a range<> representing the dimensions of the current group.
- size_t get_group_range (int dimension) const

  Return element dimension from the con stituent group range.
- range< Dimensions > get_global_range () const

  Get the local range for this work_group.
- size_t get_global_range (int dimension) const

  Return element dimension from the constituent global range.
- range< Dimensions > get_local_range () const

  Get the local range for this work_group.
- size_t get_local_range (int dimension) const

  Return element dimension from the constituent local range.
- id< Dimensions > get_offset () const

  Get the offset of the NDRange.
- size_t get_offset (int dimension) const

  Get the offset of the NDRange.

- nd_range< Dimensions > get_nd_range () const
- size_t get_linear () const

  Get a linearized version of the group ID.
- void parallel_for_work_item (std::function< void(nd_item< dimensionality >)> f) const

  Loop on the work-items inside a work-group.
- void parallel_for_work_item (std::function< void(item< dimensionality >)> f) const

  Loop on the work-items inside a work-group.

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Attributes

- id< Dimensions > group_id

  The coordinate of the group item.
- nd_range< Dimensions > ndr

  Keep a reference on the nd_range to serve potential query on it.

8.10.2.1.1 Constructor & Destructor Documentation

8.10.2.1.1.1 group() [1/3]

```
template<int Dimensions>
cl::sycl::group< Dimensions >::group (
            const nd_range< Dimensions > & ndr )  [inline]
```

Create a group from an nd_range<> with a 0 id<>

Todo This should be private since it is only used by the triSYCL implementation

Definition at line 61 of file group.hpp.

```
00061 : ndr { ndr } {}
```

8.10.2.1.1.2 group() [2/3]

```
template<int Dimensions>
cl::sycl::group< Dimensions >::group (
            const id< Dimensions > & i,
            const nd_range< Dimensions > & ndr )  [inline]
```

Create a group from an id and a nd_range<>

Todo This should be private somehow, but it is used by the validation infrastructure

Definition at line 69 of file group.hpp.

```
00069                                                    :
00070     group_id { i }, ndr { ndr } {}
```

8.10.2.1.1.3 group() [3/3]

```
template<int Dimensions>
cl::sycl::group< Dimensions >::group ( ) [default]
```

To be able to copy and assign group, use default constructors too.

Todo Make most of them protected, reserved to implementation

8.10.2.1.2 Member Function Documentation

8.10.2.1.2.1 get_global_range() [1/2]

```
template<int Dimensions>
range<Dimensions> cl::sycl::group< Dimensions >::get_global_range ( ) const [inline]
```

Get the local range for this work_group.

Definition at line 122 of file group.hpp.

```
00122                                                {
00123     return get_nd_range().get_global();
00124   }
```

8.10.2.1.2.2 get_global_range() [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_global_range (
            int dimension ) const [inline]
```

Return element dimension from the constituent global range.

Definition at line 128 of file group.hpp.

```
00128                                                  {
00129     return get_global_range()[dimension];
00130   }
```

8.10.2.1.2.3 get_group_range() [1/2]

```
template<int Dimensions>
range<Dimensions> cl::sycl::group< Dimensions >::get_group_range ( ) const [inline]
```

Return a range<> representing the dimensions of the current group.

This local range may have been provided by the programmer, or chosen by the runtime.

Todo Fix this comment and the specification

Definition at line 110 of file group.hpp.

```
00110                                                {
00111     return get_nd_range().get_group();
00112   }
```

8.10.2.1.2.4 get_group_range() [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_group_range (
            int dimension ) const  [inline]
```

Return element dimension from the con stituent group range.

Definition at line 116 of file group.hpp.

```
00116                                                {
00117     return get_group_range()[dimension];
00118   }
```

8.10.2.1.2.5 get_id() [1/2]

```
template<int Dimensions>
id<Dimensions> cl::sycl::group< Dimensions >::get_id ( ) const  [inline]
```

Return an id representing the index of the group within the nd_range for every dimension.

Definition at line 83 of file group.hpp.

Referenced by cl::sycl::detail::parallel_for_workitem().

```
00083 { return group_id; }
```

Here is the caller graph for this function:

8.10.2.1.2.6 get_id() [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_id (
            int dimension ) const  [inline]
```

Return the index of the group in the given dimension.

Definition at line 87 of file group.hpp.

References cl::sycl::group< Dimensions >::get_id().

Referenced by cl::sycl::group< Dimensions >::get_id().

```
00087 { return get_id()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.1.2.7 get_linear()

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_linear ( ) const  [inline]
```

Get a linearized version of the group ID.

Definition at line 172 of file group.hpp.

References cl::sycl::detail::linear_id().

```
00172                                {
00173     return detail::linear_id(get_group_range(),
      get_id());
00174   }
```

Here is the call graph for this function:

8.10.2.1.2.8 get_local_range() [1/2]

```
template<int Dimensions>
range<Dimensions> cl::sycl::group< Dimensions >::get_local_range ( ) const  [inline]
```

Get the local range for this work_group.

Todo Add to the specification

Definition at line 137 of file group.hpp.

Referenced by cl::sycl::detail::parallel_for_workitem().

```
00137                                          {
00138     return get_nd_range().get_local();
00139   }
```

Here is the caller graph for this function:

8.10.2.1.2.9 get_local_range() [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_local_range (
            int dimension ) const  [inline]
```

Return element dimension from the constituent local range.

Todo Add to the specification

Definition at line 146 of file group.hpp.

```
00146                                                {
00147     return get_local_range()[dimension];
00148   }
```

8.10.2.1.2.10 get_nd_range()

```
template<int Dimensions>
nd_range<Dimensions> cl::sycl::group< Dimensions >::get_nd_range ( ) const  [inline]
```

Todo Also provide this access to the current nd_range

Definition at line 166 of file group.hpp.

Referenced by cl::sycl::detail::parallel_for_workitem().

```
00166 { return ndr; }
```

Here is the caller graph for this function:

8.10.2.1.2.11 get_offset() [1/2]

```
template<int Dimensions>
id<Dimensions> cl::sycl::group< Dimensions >::get_offset ( ) const  [inline]
```

Get the offset of the NDRange.

Todo Add to the specification

Definition at line 155 of file group.hpp.

```
00155 { return get_nd_range().get_offset(); }
```

8.10.2.1.2.12 get_offset() [2/2]

```
template<int Dimensions>
size_t cl::sycl::group< Dimensions >::get_offset (
            int dimension ) const  [inline]
```

Get the offset of the NDRange.

Todo Add to the specification

Definition at line 162 of file group.hpp.

References cl::sycl::group< Dimensions >::get_offset().

Referenced by cl::sycl::group< Dimensions >::get_offset().

```
00162 { return get_offset()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.1.2.13 operator[]()

```
template<int Dimensions>
auto& cl::sycl::group< Dimensions >::operator[] (
            int dimension )  [inline]
```

Return the index of the group in the given dimension within the nd_range<>

Todo In this implementation it is not const because the group<> is written in the parallel_for iterators. To fix according to the specification

Definition at line 97 of file group.hpp.

```
00097                                   {
00098     return group_id[dimension];
00099   }
```

8.10.2.1.2.14 parallel_for_work_item() [1/2]

```
template<int Dimensions>
void cl::sycl::group< Dimensions >::parallel_for_work_item (
            std::function< void(nd_item< dimensionality >)> f ) const  [inline]
```

Loop on the work-items inside a work-group.

Todo Add this method in the specification

Definition at line 181 of file group.hpp.

References cl::sycl::detail::parallel_for_workitem().

Referenced by cl::sycl::parallel_for_work_item().

```
00182           {
00183     detail::parallel_for_workitem(*this, f);
00184   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.1.2.15 parallel_for_work_item() [2/2]

```
template<int Dimensions>
void cl::sycl::group< Dimensions >::parallel_for_work_item (
            std::function< void(item< dimensionality >)> f ) const  [inline]
```

Loop on the work-items inside a work-group.

Todo Add this method in the specification

Definition at line 191 of file group.hpp.

References i, and cl::sycl::detail::parallel_for_workitem().

```
00192           {
00193     auto item_adapter = [=] (nd_item<dimensionality> ndi) {
00194       item<dimensionality> i = ndi.get_item();
00195       f(i);
00196     };
00197     detail::parallel_for_workitem(*this, item_adapter);
00198   }
```

Here is the call graph for this function:

8.10.2.1.3 Member Data Documentation

8.10.2.1.3.1 dimensionality

```
template<int Dimensions>
constexpr auto cl::sycl::group< Dimensions >::dimensionality = Dimensions  [static]
```

Todo add this Boost::multi_array or STL concept to the specification?

Definition at line 44 of file group.hpp.

8.10.2.1.3.2 group_id

```
template<int Dimensions>
id<Dimensions> cl::sycl::group< Dimensions >::group_id  [private]
```

The coordinate of the group item.

Definition at line 49 of file group.hpp.

8.10.2.1.3.3 ndr

```
template<int Dimensions>
nd_range<Dimensions> cl::sycl::group< Dimensions >::ndr  [private]
```

Keep a reference on the nd_range to serve potential query on it.

Definition at line 52 of file group.hpp.

8.10.2.2 class cl::sycl::id

template<int Dimensions = 1>
class cl::sycl::id< Dimensions >

Define a multi-dimensional index, used for example to locate a work item.

Definition at line 31 of file id.hpp.

Inheritance diagram for cl::sycl::id< Dimensions >:

Collaboration diagram for cl::sycl::id< Dimensions >:

Public Member Functions

- id (const range< Dimensions > &range_size)

  Construct an id from the dimensions of a range.

Additional Inherited Members

8.10.2.2.1 Constructor & Destructor Documentation

8.10.2.2.1.1 id()

```
template<int Dimensions = 1>
cl::sycl::id< Dimensions >::id (
            const range< Dimensions > & range_size )  [inline]
```

Construct an id from the dimensions of a range.

Use the fact we have a constructor of a small_array from a another kind of small_array

Definition at line 45 of file id.hpp.

Referenced by cl::sycl::id< dimensionality >::id().

```
00049     : detail::small_array_123<std::size_t, id<Dimensions>, Dimensions>
00050       { range_size }
```

Here is the caller graph for this function:

8.10.2.3 class cl::sycl::item

template<int Dimensions = 1>
class cl::sycl::item< Dimensions >

A SYCL item stores information on a work-item with some more context such as the definition range and offset.

Definition at line 21 of file id.hpp.

Collaboration diagram for cl::sycl::item< Dimensions >:

Public Member Functions

- item (range< Dimensions > global_size, id< Dimensions > global_index, id< Dimensions > offset={})

  Create an item from a local size and an optional offset.
- item ()=default

  To be able to copy and assign item, use default constructors too.
- id< Dimensions > get_id () const

  Return the constituent local or global id<> representing the work-item's position in the iteration space.
- size_t get_id (int dimension) const

  Return the requested dimension of the constituent id<> representing the work-item's position in the iteration space.

- auto & operator[] (int dimension)

  Return the constituent id<> l-value representing the work-item's position in the iteration space in the given dimension.
- range< Dimensions > get_range () const

  Returns a range<> representing the dimensions of the range of possible values of the item.
- id< Dimensions > get_offset () const

  Returns an id<> representing the n-dimensional offset provided to the parallel_for and that is added by the runtime to the global-ID of each work-item, if this item represents a global range.
- size_t get_linear_id () const

  Return the linearized ID in the item's range.
- void set (id< Dimensions > Index)

  For the implementation, need to set the global index.
- void display () const

  Display the value for debugging and validation purpose.

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Attributes

- range< Dimensions > global_range
- id< Dimensions > global_index
- id< Dimensions > offset

8.10.2.3.1 Constructor & Destructor Documentation

8.10.2.3.1.1 item() [1/2]

```
template<int Dimensions = 1>
cl::sycl::item< Dimensions >::item (
            range< Dimensions > global_size,
            id< Dimensions > global_index,
            id< Dimensions > offset = {} )  [inline]
```

Create an item from a local size and an optional offset.

This constructor is used by the triSYCL implementation and the non-regression testing.

Definition at line 50 of file item.hpp.

References cl::sycl::item< Dimensions >::item().

```
00052                                {}) :
00053     global_range { global_size },
00054     global_index { global_index },
00055     offset { offset }
00056   {}
```

Here is the call graph for this function:

8.10.2.3.1.2 item() [2/2]

```
template<int Dimensions = 1>
cl::sycl::item< Dimensions >::item ( ) [default]
```

To be able to copy and assign item, use default constructors too.

Todo Make most of them protected, reserved to implementation

Referenced by cl::sycl::item< Dimensions >::item().

Here is the caller graph for this function:

8.10.2.3.2 Member Function Documentation

8.10.2.3.2.1 display()

```
template<int Dimensions = 1>
void cl::sycl::item< Dimensions >::display ( ) const [inline]
```

Display the value for debugging and validation purpose.

Definition at line 117 of file item.hpp.

References cl::sycl::detail::display_vector< range< Dimensions > >::display(), and cl::sycl::detail::display_vector< id< Dimensions > >::display().

```
00117                        {
00118     global_range.display();
00119     global_index.display();
00120     offset.display();
00121   }
```

Here is the call graph for this function:

8.10.2.3.2.2 get_id() [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::get_id ( ) const  [inline]
```

Return the constituent local or global id<> representing the work-item's position in the iteration space.

Definition at line 69 of file item.hpp.

References cl::sycl::item< Dimensions >::global_index.

Referenced by cl::sycl::item< Dimensions >::get_linear_id(), cl::sycl::id< dimensionality >::id(), and cl::sycl↩ ::accessor< DataType, Dimensions, AccessMode, Target >::operator[]().

```
00069 { return global_index; }
```

Here is the caller graph for this function:

8.10.2.3.2.3 get_id() [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::item< Dimensions >::get_id (
            int dimension ) const  [inline]
```

Return the requested dimension of the constituent id<> representing the work-item's position in the iteration space.

Definition at line 75 of file item.hpp.

References cl::sycl::item< Dimensions >::get_id().

Referenced by cl::sycl::item< Dimensions >::get_id().

```
00075 { return get_id()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.3.2.4 get_linear_id()

```
template<int Dimensions = 1>
size_t cl::sycl::item< Dimensions >::get_linear_id ( ) const  [inline]
```

Return the linearized ID in the item's range.

Computed as the flatted ID after the offset is subtracted.

Definition at line 104 of file item.hpp.

References cl::sycl::item< Dimensions >::get_id(), cl::sycl::item< Dimensions >::get_offset(), cl::sycl::item< Dimensions >::get_range(), and cl::sycl::detail::linear_id().

```
00104                                {
00105     return detail::linear_id(get_range(), get_id(),
      get_offset());
00106   }
```

Here is the call graph for this function:

8.10.2.3.2.5 get_offset()

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::get_offset ( ) const  [inline]
```

Returns an id<> representing the n-dimensional offset provided to the parallel_for and that is added by the runtime to the global-ID of each work-item, if this item represents a global range.

For an item representing a local range of where no offset was passed this will always return an id of all 0 values.

Definition at line 97 of file item.hpp.

References cl::sycl::item< Dimensions >::offset.

Referenced by cl::sycl::item< Dimensions >::get_linear_id().

```
00097 { return offset; }
```

Here is the caller graph for this function:

8.10.2.3.2.6 get_range()

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::item< Dimensions >::get_range ( ) const  [inline]
```

Returns a range<> representing the dimensions of the range of possible values of the item.

Definition at line 87 of file item.hpp.

References cl::sycl::item< Dimensions >::global_range.

Referenced by cl::sycl::item< Dimensions >::get_linear_id().

```
00087 { return global_range; }
```

Here is the caller graph for this function:

8.10.2.3.2.7 operator[]()

```
template<int Dimensions = 1>
auto& cl::sycl::item< Dimensions >::operator[] (
            int dimension )  [inline]
```

Return the constituent id<> l-value representing the work-item's position in the iteration space in the given dimension.

Definition at line 81 of file item.hpp.

```
00081 { return global_index[dimension]; }
```

8.10.2.3.2.8 set()

```
template<int Dimensions = 1>
void cl::sycl::item< Dimensions >::set (
            id< Dimensions > Index )  [inline]
```

For the implementation, need to set the global index.

Todo Move to private and add friends

Definition at line 113 of file item.hpp.

```
00113 { global_index = Index; }
```

8.10.2.3.3 Member Data Documentation

8.10.2.3.3.1 dimensionality

```
template<int Dimensions = 1>
constexpr auto cl::sycl::item< Dimensions >::dimensionality = Dimensions  [static]
```

Todo add this Boost::multi_array or STL concept to the specification?

Definition at line 35 of file item.hpp.

8.10.2.3.3.2 global_index

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::global_index  [private]
```

Definition at line 40 of file item.hpp.

Referenced by cl::sycl::item< Dimensions >::get_id().

8.10.2.3.3.3 global_range

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::item< Dimensions >::global_range  [private]
```

Definition at line 39 of file item.hpp.

Referenced by cl::sycl::item< Dimensions >::get_range().

8.10.2.3.3.4 offset

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::item< Dimensions >::offset  [private]
```

Definition at line 41 of file item.hpp.

Referenced by cl::sycl::item< Dimensions >::get_offset().

8.10.2.4 struct cl::sycl::nd_item

template<int Dimensions = 1>
struct cl::sycl::nd_item< Dimensions >

A SYCL nd_item stores information on a work-item within a work-group, with some more context such as the definition ranges.

Definition at line 33 of file nd_item.hpp.

Collaboration diagram for cl::sycl::nd_item< Dimensions >:

Public Member Functions

- nd_item (nd_range< Dimensions > ndr)

  Create an empty nd_item<> from an nd_range<>
- nd_item (id< Dimensions > global_index, nd_range< Dimensions > ndr)

  Create a full nd_item.
- nd_item ()=default

  To be able to copy and assign nd_item, use default constructors too.
- id< Dimensions > get_global () const

  Return the constituent global id representing the work-item's position in the global iteration space.
- size_t get_global (int dimension) const

  Return the constituent element of the global id representing the work-item's position in the global iteration space in the given dimension.
- size_t get_global_linear_id () const

  Return the flattened id of the current work-item after subtracting the offset.
- id< Dimensions > get_local () const

  Return the constituent local id representing the work-item's position within the current work-group.
- size_t get_local (int dimension) const

  Return the constituent element of the local id representing the work-item's position within the current work-group in the given dimension.
- size_t get_local_linear_id () const

  Return the flattened id of the current work-item within the current work-group.
- id< Dimensions > get_group () const

  Return the constituent group group representing the work-group's position within the overall nd_range.
- size_t get_group (int dimension) const

  Return the constituent element of the group id representing the work-group;s position within the overall nd_range in the given dimension.

- size_t get_group_linear_id () const

  Return the flattened id of the current work-group.
- id< Dimensions > get_num_groups () const

  Return the number of groups in the nd_range.
- size_t get_num_groups (int dimension) const

  Return the number of groups for dimension in the nd_range.
- range< Dimensions > get_global_range () const

  Return a range<> representing the dimensions of the nd_range<>
- range< Dimensions > get_local_range () const

  Return a range<> representing the dimensions of the current work-group.
- id< Dimensions > get_offset () const

  Return an id<> representing the n-dimensional offset provided to the constructor of the nd_range<> and that is added by the runtime to the global-ID of each work-item.
- nd_range< Dimensions > get_nd_range () const

  Return the nd_range<> of the current execution.
- item< Dimensions > get_item () const

  Allows projection down to an item.
- void barrier (access::fence_space flag=access::fence_space::global_and_local) const

  Execute a barrier with memory ordering on the local address space, global address space or both based on the value of flag.
- void set_local (id< Dimensions > Index)
- void set_global (id< Dimensions > Index)

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Attributes

- id< Dimensions > global_index
- id< Dimensions > local_index
- nd_range< Dimensions > ND_range

8.10.2.4.1 Constructor & Destructor Documentation

8.10.2.4.1.1 nd_item() [1/3]

```
template<int Dimensions = 1>
cl::sycl::nd_item< Dimensions >::nd_item (
            nd_range< Dimensions > ndr )  [inline]
```

Create an empty nd_item<> from an nd_range<>

Todo This is for the triSYCL implementation which is expected to call set_global() and set_local() later. This should be hidden to the user.

Definition at line 54 of file nd_item.hpp.

```
00054 : ND_range { ndr } {}
```

8.10.2.4.1.2 nd_item() [2/3]

```
template<int Dimensions = 1>
cl::sycl::nd_item< Dimensions >::nd_item (
            id< Dimensions > global_index,
            nd_range< Dimensions > ndr )  [inline]
```

Create a full nd_item.

Todo This is for validation purpose. Hide this to the programmer somehow

Definition at line 62 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::nd_item().

```
00063                                   :
00064     global_index { global_index },
00065     // Compute the local index using the offset and the group size
00066     local_index
00067       { (global_index - ndr.get_offset())%id<Dimensions> { ndr.get_local() } },
00068     ND_range { ndr }
00069   {}
```

Here is the call graph for this function:

8.10.2.4.1.3 nd_item() [3/3]

```
template<int Dimensions = 1>
cl::sycl::nd_item< Dimensions >::nd_item ( )  [default]
```

To be able to copy and assign nd_item, use default constructors too.

Todo Make most of them protected, reserved to implementation

Referenced by cl::sycl::nd_item< Dimensions >::nd_item().

Here is the caller graph for this function:

8.10.2.4.2 Member Function Documentation

8.10.2.4.2.1 barrier()

```
template<int Dimensions = 1>
void cl::sycl::nd_item< Dimensions >::barrier (
            access::fence_space flag = access::fence_space::global_and_local ) const  [inline]
```

Execute a barrier with memory ordering on the local address space, global address space or both based on the value of flag.

The current work-item will wait at the barrier until all work-items in the current work-group have reached the barrier.

In addition, the barrier performs a fence operation ensuring that all memory accesses in the specified address space issued before the barrier complete before those issued after the barrier

Definition at line 199 of file nd_item.hpp.

References cl::sycl::detail::unimplemented().

```
00200                                                       {
00201 #if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
00202     /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
00203        work-item of the work-group */
00204 #pragma omp barrier
00205 #else
00206     // \todo To be implemented efficiently otherwise
00207     detail::unimplemented();
00208 #endif
00209   }
```

Here is the call graph for this function:

8.10.2.4.2.2 get_global() [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_global ( ) const  [inline]
```

Return the constituent global id representing the work-item's position in the global iteration space.

Definition at line 82 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::global_index.

Referenced by cl::sycl::nd_item< Dimensions >::get_global_linear_id(), cl::sycl::nd_item< Dimensions >::get_group(), cl::sycl::nd_item< Dimensions >::get_item(), cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::operator[](), cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >::operator[](), and cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::operator[]().

```
00082 { return global_index; }
```

Here is the caller graph for this function:

8.10.2.4.2.3 get_global() [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_global (
            int dimension ) const  [inline]
```

Return the constituent element of the global id representing the work-item's position in the global iteration space in the given dimension.

Definition at line 89 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_global().

Referenced by cl::sycl::nd_item< Dimensions >::get_global().

```
00089 { return get_global()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.4 get_global_linear_id()

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_global_linear_id ( ) const  [inline]
```

Return the flattened id of the current work-item after subtracting the offset.

Definition at line 95 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_global(), cl::sycl::nd_item< Dimensions >::get_global_range(), cl::sycl::nd_item< Dimensions >::get_offset(), and cl::sycl::detail::linear_id().

```
00095                                    {
00096     return detail::linear_id(get_global_range(),
      get_global(), get_offset());
00097   }
```

Here is the call graph for this function:

8.10.2.4.2.5 get_global_range()

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_item< Dimensions >::get_global_range ( ) const  [inline]
```

Return a range<> representing the dimensions of the nd_range<>

Definition at line 158 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_nd_range().

Referenced by cl::sycl::nd_item< Dimensions >::get_global_linear_id(), and cl::sycl::nd_item< Dimensions >↩ ::get_item().

```
00158                                                 {
00159     return get_nd_range().get_global();
00160   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.6 get_group() [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_group ( ) const  [inline]
```

Return the constituent group group representing the work-group's position within the overall nd_range.

Definition at line 124 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_global(), and cl::sycl::nd_item< Dimensions >::get_local_range().

Referenced by cl::sycl::nd_item< Dimensions >::get_group(), and cl::sycl::nd_item< Dimensions >::get_group_linear_id().

```
00124                                   {
00125     /* Convert get_local_range() to an id<> to remove ambiguity into using
00126        implicit conversion either from range<> to id<> or the opposite */
00127     return get_global()/id<Dimensions> { get_local_range() };
00128   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.7 get_group() [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_group (
            int dimension ) const  [inline]
```

Return the constituent element of the group id representing the work-group;s position within the overall nd_range in the given dimension.

Definition at line 135 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_group().

```
00135                                       {
00136     return get_group()[dimension];
00137   }
```

Here is the call graph for this function:

8.10.2.4.2.8 get_group_linear_id()

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_group_linear_id ( ) const  [inline]
```

Return the flattened id of the current work-group.

Definition at line 141 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_group(), cl::sycl::nd_item< Dimensions >::get_num_groups(), and cl::sycl::detail::linear_id().

```
00141                                    {
00142     return detail::linear_id(get_num_groups(),
      get_group());
00143   }
```

Here is the call graph for this function:

8.10.2.4.2.9 get_item()

```
template<int Dimensions = 1>
item<Dimensions> cl::sycl::nd_item< Dimensions >::get_item ( ) const  [inline]
```

Allows projection down to an item.

Todo Add to the specification

Definition at line 184 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_global(), cl::sycl::nd_item< Dimensions >::get_global_range(), and cl::sycl::nd_item< Dimensions >::get_offset().

```
00184                                    {
00185     return { get_global_range(), get_global(),
      get_offset() };
00186   }
```

Here is the call graph for this function:

8.10.2.4.2.10 get_local() [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_local ( ) const  [inline]
```

Return the constituent local id representing the work-item's position within the current work-group.

Definition at line 103 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::local_index.

Referenced by cl::sycl::nd_item< Dimensions >::get_local_linear_id().

```
00103 { return local_index; }
```

Here is the caller graph for this function:

8.10.2.4.2.11 get_local() [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_local (
            int dimension ) const [inline]
```

Return the constituent element of the local id representing the work-item's position within the current work-group in the given dimension.

Definition at line 110 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_local().

Referenced by cl::sycl::nd_item< Dimensions >::get_local().

```
00110 { return get_local()[dimension]; }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.12 get_local_linear_id()

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_local_linear_id ( ) const  [inline]
```

Return the flattened id of the current work-item within the current work-group.

Definition at line 116 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_local(), cl::sycl::nd_item< Dimensions >::get_local_range(), and cl::sycl::detail::linear_id().

```
00116                                       {
00117     return detail::linear_id(get_local_range(),
      get_local());
00118   }
```

Here is the call graph for this function:

8.10.2.4.2.13 get_local_range()

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_item< Dimensions >::get_local_range ( ) const  [inline]
```

Return a range<> representing the dimensions of the current work-group.

Definition at line 164 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_nd_range().

Referenced by cl::sycl::nd_item< Dimensions >::get_group(), and cl::sycl::nd_item< Dimensions >::get_local_↩linear_id().

```
00164                                           {
00165     return get_nd_range().get_local();
00166   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.14 get_nd_range()

```
template<int Dimensions = 1>
nd_range<Dimensions> cl::sycl::nd_item< Dimensions >::get_nd_range ( ) const  [inline]
```

Return the nd_range<> of the current execution.

Definition at line 177 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::ND_range.

Referenced by cl::sycl::nd_item< Dimensions >::get_global_range(), cl::sycl::nd_item< Dimensions >::get_local↩_range(), cl::sycl::nd_item< Dimensions >::get_num_groups(), and cl::sycl::nd_item< Dimensions >::get_offset().

```
00177 { return ND_range; }
```

Here is the caller graph for this function:

8.10.2.4.2.15 get_num_groups() [1/2]

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_num_groups ( ) const  [inline]
```

Return the number of groups in the nd_range.

Definition at line 147 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_nd_range().

Referenced by cl::sycl::nd_item< Dimensions >::get_group_linear_id(), and cl::sycl::nd_item< Dimensions >↩ ::get_num_groups().

```
00147                                       {
00148     return get_nd_range().get_group();
00149   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.16 get_num_groups() [2/2]

```
template<int Dimensions = 1>
size_t cl::sycl::nd_item< Dimensions >::get_num_groups (
            int dimension ) const [inline]
```

Return the number of groups for dimension in the nd_range.

Definition at line 152 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_num_groups().

```
00152                                                {
00153       return get_num_groups()[dimension];
00154   }
```

Here is the call graph for this function:

8.10.2.4.2.17 get_offset()

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::get_offset ( ) const  [inline]
```

Return an id<> representing the n-dimensional offset provided to the constructor of the nd_range<> and that is added by the runtime to the global-ID of each work-item.

Definition at line 173 of file nd_item.hpp.

References cl::sycl::nd_item< Dimensions >::get_nd_range().

Referenced by cl::sycl::nd_item< Dimensions >::get_global_linear_id(), and cl::sycl::nd_item< Dimensions >↩ ::get_item().

```
00173 { return get_nd_range().get_offset(); }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.2.4.2.18 set_global()

```
template<int Dimensions = 1>
void cl::sycl::nd_item< Dimensions >::set_global (
            id< Dimensions > Index )  [inline]
```

Definition at line 217 of file nd_item.hpp.

```
00217 { global_index = Index; }
```

8.10.2.4.2.19 set_local()

```
template<int Dimensions = 1>
void cl::sycl::nd_item< Dimensions >::set_local (
            id< Dimensions > Index )  [inline]
```

Definition at line 213 of file nd_item.hpp.

```
00213 { local_index = Index; }
```

8.10.2.4.3 Member Data Documentation

8.10.2.4.3.1 dimensionality

```
template<int Dimensions = 1>
constexpr auto cl::sycl::nd_item< Dimensions >::dimensionality = Dimensions  [static]
```

Todo add this Boost::multi_array or STL concept to the specification?

Definition at line 36 of file nd_item.hpp.

8.10.2.4.3.2 global_index

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::global_index  [private]
```

Definition at line 40 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< Dimensions >::get_global().

8.10.2.4.3.3 local_index

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_item< Dimensions >::local_index  [private]
```

Definition at line 43 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< Dimensions >::get_local().

8.10.2.4.3.4 ND_range

```
template<int Dimensions = 1>
nd_range<Dimensions> cl::sycl::nd_item< Dimensions >::ND_range  [private]
```

Definition at line 44 of file nd_item.hpp.

Referenced by cl::sycl::nd_item< Dimensions >::get_nd_range().

8.10.2.5 struct cl::sycl::nd_range

template<int Dimensions = 1>
struct cl::sycl::nd_range< Dimensions >

A ND-range, made by a global and local range, to specify work-group and work-item organization.

The local offset is used to translate the iteration space origin if needed.

Todo add copy constructors in the specification

Definition at line 33 of file nd_range.hpp.

Collaboration diagram for cl::sycl::nd_range< Dimensions >:

Public Member Functions

- nd_range (range< Dimensions > global_size, range< Dimensions > local_size, id< Dimensions > offset={})

  Construct a ND-range with all the details available in OpenCL.
- range< Dimensions > get_global () const

  Get the global iteration space range.
- range< Dimensions > get_local () const

  Get the local part of the iteration space range.
- auto get_group () const

  Get the range of work-groups needed to run this ND-range.
- id< Dimensions > get_offset () const
- void display () const

  Display the value for debugging and validation purpose.

Static Public Attributes

- static constexpr auto dimensionality = Dimensions

Private Attributes

- range< dimensionality > global_range
- range< dimensionality > local_range
- id< dimensionality > offset

8.10.2.5.1 Constructor & Destructor Documentation

8.10.2.5.1.1 nd_range()

```
template<int Dimensions = 1>
cl::sycl::nd_range< Dimensions >::nd_range (
            range< Dimensions > global_size,
            range< Dimensions > local_size,
            id< Dimensions > offset = {} )  [inline]
```

Construct a ND-range with all the details available in OpenCL.

By default use a zero offset, that is iterations start at 0

Definition at line 50 of file nd_range.hpp.

```
00052                                   {}) :
00053     global_range { global_size }, local_range { local_size },
      offset { offset }
00054   { }
```

8.10.2.5.2 Member Function Documentation

8.10.2.5.2.1 display()

```
template<int Dimensions = 1>
void cl::sycl::nd_range< Dimensions >::display ( ) const  [inline]
```

Display the value for debugging and validation purpose.

Definition at line 80 of file nd_range.hpp.

References cl::sycl::detail::display_vector< T >::display().

```
00080                       {
00081     global_range.display();
00082     local_range.display();
00083     offset.display();
00084   }
```

Here is the call graph for this function:

8.10.2.5.2.2 get_global()

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_range< Dimensions >::get_global ( ) const  [inline]
```

Get the global iteration space range.

Definition at line 58 of file nd_range.hpp.

References cl::sycl::nd_range< Dimensions >::global_range.

```
00058 { return global_range; }
```

8.10.2.5.2.3 get_group()

```
template<int Dimensions = 1>
auto cl::sycl::nd_range< Dimensions >::get_group ( ) const  [inline]
```

Get the range of work-groups needed to run this ND-range.

Definition at line 66 of file nd_range.hpp.

Referenced by cl::sycl::detail::parallel_for(), and cl::sycl::detail::parallel_for_workgroup().

```
00066                          {
00067     /* This is basically global_range/local_range, round up to the
00068        next integer, in case the global range is not a multiple of the
00069        local range. Note this is a motivating example to build a range
00070        from a scalar with a broadcasting constructor. */
00071     return (global_range + local_range - range<Dimensions>{ 1 })/
      local_range;
00072   }
```

Here is the caller graph for this function:

8.10.2.5.2.4 get_local()

```
template<int Dimensions = 1>
range<Dimensions> cl::sycl::nd_range< Dimensions >::get_local ( ) const  [inline]
```

Get the local part of the iteration space range.

Definition at line 62 of file nd_range.hpp.

References cl::sycl::nd_range< Dimensions >::local_range.

Referenced by cl::sycl::detail::parallel_for().

```
00062 { return local_range; }
```

Here is the caller graph for this function:

8.10.2.5.2.5 get_offset()

```
template<int Dimensions = 1>
id<Dimensions> cl::sycl::nd_range< Dimensions >::get_offset ( ) const  [inline]
```

Todo get_offset() is lacking in the specification

Definition at line 76 of file nd_range.hpp.

References cl::sycl::nd_range< Dimensions >::offset.

```
00076 { return offset; }
```

8.10.2.5.3 Member Data Documentation

8.10.2.5.3.1 dimensionality

```
template<int Dimensions = 1>
constexpr auto cl::sycl::nd_range< Dimensions >::dimensionality = Dimensions  [static]
```

Todo add this Boost::multi_array or STL concept to the specification?

Definition at line 36 of file nd_range.hpp.

8.10.2.5.3.2 global_range

```
template<int Dimensions = 1>
range<dimensionality> cl::sycl::nd_range< Dimensions >::global_range  [private]
```

Definition at line 40 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< Dimensions >::get_global().

8.10.2.5.3.3 local_range

```
template<int Dimensions = 1>
range<dimensionality> cl::sycl::nd_range< Dimensions >::local_range  [private]
```

Definition at line 41 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< Dimensions >::get_local().

8.10.2.5.3.4 offset

```
template<int Dimensions = 1>
id<dimensionality> cl::sycl::nd_range< Dimensions >::offset  [private]
```

Definition at line 42 of file nd_range.hpp.

Referenced by cl::sycl::nd_range< Dimensions >::get_offset().

8.10.2.6 struct cl::sycl::detail::parallel_for_iterate

template<std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

A recursive multi-dimensional iterator that ends up calling f.

The iteration order may be changed later.

Since partial specialization of function template is not possible in C++14, use a class template instead with everything in the constructor.

Definition at line 50 of file parallelism.hpp.

Public Member Functions

- parallel_for_iterate (Range r, ParallelForFunctor &f, Id &index)

8.10.2.6.1 Constructor & Destructor Documentation

8.10.2.6.1.1 parallel_for_iterate()

```
template<std::size_t level, typename Range , typename ParallelForFunctor , typename Id >
cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >::parallel_for↩
_iterate (
            Range r,
            ParallelForFunctor & f,
            Id & index )  [inline]
```

Definition at line 51 of file parallelism.hpp.

```
00051                                                                   {
00052     for (boost::multi_array_types::index _sycl_index = 0,
00053            _sycl_end = r[Range::dimensionality - level];
00054          _sycl_index < _sycl_end;
00055          _sycl_index++) {
00056       // Set the current value of the index for this dimension
00057       index[Range::dimensionality - level] = _sycl_index;
00058       // Iterate further on lower dimensions
00059       parallel_for_iterate<level - 1,
00060                            Range,
00061                            ParallelForFunctor,
00062                            Id> { r, f, index };
00063     }
00064   }
```

8.10.2.7 struct cl::sycl::detail::parallel_OpenMP_for_iterate

template<std::size_t level, typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >

A top-level recursive multi-dimensional iterator variant using OpenMP.

Only the top-level loop uses OpenMP and goes on with the normal recursive multi-dimensional.

Definition at line 77 of file parallelism.hpp.

Public Member Functions

- parallel_OpenMP_for_iterate (Range r, ParallelForFunctor &f)

8.10.2.7.1 Constructor & Destructor Documentation

8.10.2.7.1.1 parallel_OpenMP_for_iterate()

```
template<std::size_t level, typename Range , typename ParallelForFunctor , typename Id >
cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >::parallel↩
_OpenMP_for_iterate (
            Range r,
            ParallelForFunctor & f )  [inline]
```

Definition at line 78 of file parallelism.hpp.

```
00078                                                                   {
00079     // Create the OpenMP threads before the for-loop to avoid creating an
00080     // index in each iteration
00081 #pragma omp parallel
00082     {
00083       // Allocate an OpenMP thread-local index
00084       Id index;
00085       // Make a simple loop end condition for OpenMP
00086       boost::multi_array_types::index _sycl_end =
00087         r[Range::dimensionality - level];
00088       /* Distribute the iterations on the OpenMP threads. Some OpenMP
00089          "collapse" could be useful for small iteration space, but it
00090          would need some template specialization to have real contiguous
00091          loop nests */
00092 #pragma omp for
00093       for (boost::multi_array_types::index _sycl_index = 0;
00094            _sycl_index < _sycl_end;
00095            _sycl_index++) {
00096         // Set the current value of the index for this dimension
00097         index[Range::dimensionality - level] = _sycl_index;
00098         // Iterate further on lower dimensions
00099         parallel_for_iterate<level - 1,
00100                              Range,
00101                              ParallelForFunctor,
00102                              Id> { r, f, index };
00103       }
00104     }
00105   }
```

8.10.2.8 struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

template<typename Range, typename ParallelForFunctor, typename Id>
struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id.

Definition at line 112 of file parallelism.hpp.

Public Member Functions

- parallel_for_iterate (Range r, ParallelForFunctor &f, Id &index)

8.10.2.8.1 Constructor & Destructor Documentation

8.10.2.8.1.1 parallel_for_iterate()

```
template<typename Range , typename ParallelForFunctor , typename Id >
cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >::parallel_for_↩
iterate (
            Range r,
            ParallelForFunctor & f,
            Id & index )  [inline]
```

Definition at line 113 of file parallelism.hpp.

```
00113                                                                    {
00114     f(index);
00115   }
```

8.10.2.9 class cl::sycl::range

template<int Dimensions = 1>
class cl::sycl::range< Dimensions >

A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes.

Todo use std::size_t Dimensions instead of int Dimensions in the specification?

Todo add to the specification this default parameter value?

Todo add to the specification some way to specify an offset?

Definition at line 34 of file range.hpp.

Inheritance diagram for cl::sycl::range< Dimensions >:

Collaboration diagram for cl::sycl::range< Dimensions >:

Public Member Functions

- size_t get_count () const

  Return the number of elements in the range.

Additional Inherited Members

8.10.2.9.1 Member Function Documentation

8.10.2.9.1.1 get_count()

```
template<int Dimensions = 1>
size_t cl::sycl::range< Dimensions >::get_count ( ) const  [inline]
```

Return the number of elements in the range.

Todo Give back size() its real meaning in the specification

Todo add this method to the specification

Definition at line 53 of file range.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >::allocate_accessor(), and cl::sycl::detail::buffer< T, Dimensions >::allocate_buffer().

```
00053                           {
00054     // Return the product of the sizes in each dimension
00055     return std::accumulate(this->cbegin(),
00056                            this->cend(),
00057                            1,
00058                            std::multiplies<size_t> {});
00059   }
```

Here is the caller graph for this function:

8.10.3 Function Documentation

8.10.3.1 make_id() [1/4]

```
auto cl::sycl::make_id (
            id< 1 > i )  [inline]
```

#include <include/CL/sycl/id.hpp>

Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.

Cannot use a template on the number of dimensions because the implicit conversion would not be tried.

Definition at line 71 of file id.hpp.

References i.

```
00071 { return i; }
```

8.10.3.2 make_id() [2/4]

```
auto cl::sycl::make_id (
            id< 2 > i )  [inline]
```

#include <include/CL/sycl/id.hpp>

Definition at line 72 of file id.hpp.

References i.

```
00072 { return i; }
```

8.10.3.3 make_id() [3/4]

```
auto cl::sycl::make_id (
            id< 3 > i )  [inline]
```

#include <include/CL/sycl/id.hpp>

Definition at line 73 of file id.hpp.

References i.

```
00073 { return i; }
```

8.10.3.4 make_id() [4/4]

```
template<typename...  BasicType>
auto cl::sycl::make_id (
            BasicType...  Args )
```

#include <include/CL/sycl/id.hpp>

Construct an id<> from a function call with arguments, like make_id(1, 2, 3)

Definition at line 79 of file id.hpp.

```
00079                                {
00080   // Call constructor directly to allow narrowing
00081   return id<sizeof...(Args)>(Args...);
00082 }
```

8.10.3.5 make_range() [1/4]

```
auto cl::sycl::make_range (
            range< 1 > r )  [inline]
```

#include <include/CL/sycl/range.hpp>

Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.

Cannot use a template on the number of dimensions because the implicit conversion would not be tried.

Definition at line 69 of file range.hpp.

```
00069 { return r; }
```

8.10.3.6 make_range() [2/4]

```
auto cl::sycl::make_range (
            range< 2 > r )  [inline]
```

```
#include <include/CL/sycl/range.hpp>
```

Definition at line 70 of file range.hpp.

```
00070 { return r; }
```

8.10.3.7 make_range() [3/4]

```
auto cl::sycl::make_range (
            range< 3 > r )  [inline]
```

```
#include <include/CL/sycl/range.hpp>
```

Definition at line 71 of file range.hpp.

```
00071 { return r; }
```

8.10.3.8 make_range() [4/4]

```
template<typename...  BasicType>
auto cl::sycl::make_range (
            BasicType...  Args )
```

```
#include <include/CL/sycl/range.hpp>
```

Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

Definition at line 78 of file range.hpp.

```
00078                                   {
00079   // Call constructor directly to allow narrowing
00080   return range<sizeof...(Args)>(Args...);
00081 }
```

8.10.3.9 parallel_for() [1/4]

```
template<int Dimensions = 1, typename ParallelForFunctor , typename Id >
void cl::sycl::detail::parallel_for (
            range< Dimensions > r,
            ParallelForFunctor f,
            Id  )
```

#include <include/CL/sycl/parallelism/detail/parallelism.hpp>

Implementation of a data parallel computation with parallelism specified at launch time by a range<>.

Kernel index is id or int.

This implementation use OpenMP 3 if compiled with the right flag.

Definition at line 125 of file parallelism.hpp.

Referenced by cl::sycl::detail::parallel_for(), cl::sycl::handler::parallel_for(), and cl::sycl::detail::parallel_for_global↩_offset().

```
00127                     {
00128 #ifdef _OPENMP
00129   // Use OpenMP for the top loop level
00130   parallel_OpenMP_for_iterate<Dimensions,
00131                               range<Dimensions>,
00132                               ParallelForFunctor,
00133                               id<Dimensions>> { r, f };
00134 #else
00135   // In a sequential execution there is only one index processed at a time
00136   id<Dimensions> index;
00137   parallel_for_iterate<Dimensions,
00138                        range<Dimensions>,
00139                        ParallelForFunctor,
00140                        id<Dimensions>> { r, f, index };
00141 #endif
00142 }
```

Here is the caller graph for this function:

8.10.3.10 parallel_for() [2/4]

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::detail::parallel_for (
            range< Dimensions > r,
            ParallelForFunctor f,
            item< Dimensions >  )
```

#include <include/CL/sycl/parallelism/detail/parallelism.hpp>

Implementation of a data parallel computation with parallelism specified at launch time by a range<>.

Kernel index is item.

This implementation use OpenMP 3 if compiled with the right flag.

Definition at line 151 of file parallelism.hpp.

```
00153                                   {
00154   auto reconstruct_item = [&] (id<Dimensions> l) {
00155     // Reconstruct the global item
00156     item<Dimensions> index { r, l };
00157     // Call the user kernel with the item<> instead of the id<>
00158     f(index);
00159   };
00160 #ifdef _OPENMP
00161   // Use OpenMP for the top loop level
00162   parallel_OpenMP_for_iterate<Dimensions,
00163                               range<Dimensions>,
00164                               decltype(reconstruct_item),
00165                               id<Dimensions>> { r, reconstruct_item };
00166 #else
00167   // In a sequential execution there is only one index processed at a time
00168   id<Dimensions> index;
00169   parallel_for_iterate<Dimensions,
00170                        range<Dimensions>,
00171                        decltype(reconstruct_item),
00172                        id<Dimensions>> { r, reconstruct_item, index };
00173 #endif
00174 }
```

8.10.3.11 parallel_for() [3/4]

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::detail::parallel_for (
            range< Dimensions > r,
            ParallelForFunctor f )
```

#include <include/CL/sycl/parallelism/detail/parallelism.hpp>

Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.

Definition at line 182 of file parallelism.hpp.

References cl::sycl::detail::parallel_for().

```
00182                                                           {
00183   using mf_t  = decltype(std::mem_fn(&ParallelForFunctor::operator()));
00184   using arg_t = typename mf_t::second_argument_type;
00185   parallel_for(r,f,arg_t{});
00186 }
```

Here is the call graph for this function:

8.10.3.12 parallel_for() [4/4]

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::detail::parallel_for (
            nd_range< Dimensions > r,
            ParallelForFunctor f )
```

#include <include/CL/sycl/parallelism/detail/parallelism.hpp>

Implement a variation of parallel_for to take into account a nd_range<>

Todo Add an OpenMP implementation

Todo Deal with incomplete work-groups

Todo Implement with parallel_for_workgroup()/parallel_for_workitem()

Definition at line 217 of file parallelism.hpp.

References cl::sycl::nd_range< Dimensions >::get_group(), cl::sycl::nd_range< Dimensions >::get_local(), and cl::sycl::detail::parallel_for_workitem().

```
00218                                       {
00219   // To iterate on the work-group
00220   id<Dimensions> group;
00221   range<Dimensions> group_range = r.get_group();
00222
00223 #ifdef _OPENMP
00224
00225   auto iterate_in_work_group = [&] (id<Dimensions> g) {
00226     //group.display();
00227
00228     // Then iterate on the local work-groups
00229     cl::sycl::group<Dimensions> wg {g, r};
00230     parallel_for_workitem<Dimensions,
00231                           decltype(f)>(wg, f);
00232   };
00233
00234 #else
00235
00236   // In a sequential execution there is only one index processed at a time
00237   nd_item<Dimensions> index { r };
00238
00239   // To iterate on the local work-item
00240   id<Dimensions> local;
00241   range<Dimensions> local_range = r.get_local();
00242
```

```
  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // Upgrade local_range to an id<> so that we can * with the group (an id<>)
    index.set_global(local + id<Dimensions>(local_range)*group);
    // Call the user kernel at last
    f(index);
  };

  /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
     into another functor that iterates inside the work-group and then
     calls f */
  auto iterate_in_work_group = [&] (id<Dimensions> g) {
    //group.display();
    // Then iterate on the local work-groups
    parallel_for_iterate<Dimensions,
                         range<Dimensions>,
                         decltype(reconstruct_item),
                         id<Dimensions>> { local_range,
                                           reconstruct_item,
                                           local };
  };

#endif

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(iterate_in_work_group),
                       id<Dimensions>> { group_range,
                                         iterate_in_work_group,
                                         group };
}
```

Here is the call graph for this function:

8.10.3.13 parallel_for_global_offset()

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::detail::parallel_for_global_offset (
            range< Dimensions > global_size,
            id< Dimensions > offset,
            ParallelForFunctor f )
```

```
#include <include/CL/sycl/parallelism/detail/parallelism.hpp>
```

Implementation of parallel_for with a range<> and an offset.

Definition at line 191 of file parallelism.hpp.

References offset, and cl::sycl::detail::parallel_for().

```
00193                                                    {
00194   // Reconstruct the item from its id<> and its offset
00195   auto reconstruct_item = [&] (id<Dimensions> l) {
00196     // Reconstruct the global item
00197     item<Dimensions> index { global_size, l + offset, offset };
00198     // Call the user kernel with the item<> instead of the id<>
00199     f(index);
00200   };
00201
00202   // First iterate on all the work-groups
00203   parallel_for(global_size, reconstruct_item);
00204 }
```

Here is the call graph for this function:

8.10.3.14 parallel_for_work_item()

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::parallel_for_work_item (
            const group< Dimensions > & g,
            ParallelForFunctor f )
```

```
#include <include/CL/sycl/parallelism.hpp>
```

SYCL parallel_for version that allows a Program object to be specified.

Todo To be implemented

Loop on the work-items inside a work-group

Todo Deprecate this function in the specification to use instead the group method

Definition at line 39 of file parallelism.hpp.

References cl::sycl::group< Dimensions >::parallel_for_work_item().

```
00040                                                   {
00041     g.parallel_for_work_item(f);
00042   }
```

Here is the call graph for this function:

8.10.3.15 parallel_for_workgroup()

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::detail::parallel_for_workgroup (
            nd_range< Dimensions > r,
            ParallelForFunctor f )
```

```
#include <include/CL/sycl/parallelism/detail/parallelism.hpp>
```

Implement the loop on the work-groups.

Definition at line 282 of file parallelism.hpp.

References cl::sycl::nd_range< Dimensions >::get_group().

Referenced by cl::sycl::handler::parallel_for_work_group().

```
00283                                                   {
00284   // In a sequential execution there is only one index processed at a time
00285   group<Dimensions> g { r };
00286
00287   // First iterate on all the work-groups
00288   parallel_for_iterate<Dimensions,
00289                        range<Dimensions>,
00290                        ParallelForFunctor,
00291                        group<Dimensions>> {
00292     r.get_group(),
00293     f,
00294     g };
00295 }
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.10.3.16 parallel_for_workitem()

```
template<int Dimensions = 1, typename ParallelForFunctor >
void cl::sycl::detail::parallel_for_workitem (
            const group< Dimensions > & g,
            ParallelForFunctor f )
```

```
#include <include/CL/sycl/group.hpp>
```

Implement the loop on the work-items inside a work-group.

Todo Better type the functor

Definition at line 303 of file parallelism.hpp.

References cl::sycl::detail::small_array< std::size_t, range< Dimensions >, Dims >::get(), cl::sycl::group< Dimensions >::get_id(), cl::sycl::group< Dimensions >::get_local_range(), cl::sycl::group< Dimensions >::get_↩nd_range(), and i.

Referenced by cl::sycl::detail::parallel_for(), and cl::sycl::group< Dimensions >::parallel_for_work_item().

```
                                                {
#if defined(_OPENMP) && (!defined(TRISYCL_NO_BARRIER) && !defined(_MSC_VER))
  /* To implement barriers With OpenMP, one thread is created for each
     work-item in the group and thus an OpenMP barrier has the same effect
     of an OpenCL barrier executed by the work-items in a workgroup

     The issue is that the parallel_for_workitem() execution is slow even
     when nd_item::barrier() is not used

     \todo Simplify by just using omp parallel for collapse
  */

  range<Dimensions> l_r = g.get_nd_range().get_local();
  auto tot = l_r.get(0);
  for (int i = 1; i < (int) Dimensions; ++i){
    tot *= l_r.get(i);
  }
#pragma omp parallel num_threads(tot)
  {
    nd_item<Dimensions> index { g.get_nd_range() };
    id<Dimensions> local; // to initialize correctly
#pragma omp for nowait
    for (std::size_t th_id = 0; th_id < tot; ++th_id) {
      if (Dimensions == 1) {
        local[0] = th_id;
      } else if (Dimensions == 2) {
        local[0] = th_id / l_r.get(1);
        local[1] = th_id % l_r.get(1);
      } else if (Dimensions == 3) {
        local[0] = th_id / (l_r.get(1)*l_r.get(2));
        local[1] = (th_id / l_r.get(2)) % l_r.get(1);
        local[2] = th_id % l_r.get(2);
      }
      index.set_local(local);
      index.set_global(local + id<Dimensions>(l_r)*g.get_id());
      f(index);
    }
  }
#else
  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { g.get_nd_range() };
  // To iterate on the local work-item
  id<Dimensions> local;

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    //l.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // \todo Some strength reduction here
    index.set_global(local + id<Dimensions>(g.get_local_range())*g.get_id());
    // Call the user kernel at last
    f(index);
  };

  // Then iterate on all the work-items of the work-group
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(reconstruct_item),
                       id<Dimensions>> {
    g.get_local_range(),
    reconstruct_item,
    local };
#endif
}
```

Here is the call graph for this function:

Here is the caller graph for this function:

8.11 Vector types in SYCL

Classes

- class cl::sycl::vec< DataType, NumElements >

  Small OpenCL vector class. More...

Macros

- #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) using type##size = vec<actual_type, size>;

  A macro to define type alias, such as for type=uchar, size=4 and actual_type=unsigned char, uchar4 is equivalent to vec<unsigned char, 4>
- #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)

  Declare the vector types of a type for all the sizes.

8.11.1 Detailed Description

8.11.2 Class Documentation

8.11.2.1 class cl::sycl::vec

template<typename DataType, size_t NumElements>
class cl::sycl::vec< DataType, NumElements >

Small OpenCL vector class.

Todo add [] operator

Todo add iterators on elements, with begin() and end()

Todo having vec<> sub-classing array<> instead would solve the previous issues

Todo move the implementation elsewhere

Todo simplify the helpers by removing some template types since there are now inside the vec<> class.

Todo rename in the specification element_type to value_type

Definition at line 42 of file vec.hpp.

Inheritance diagram for cl::sycl::vec< DataType, NumElements >:

Collaboration diagram for cl::sycl::vec< DataType, NumElements >:

Public Member Functions

- template<typename... Types>
  vec (const Types... args)

  Construct a vec from anything from a scalar (to initialize all the elements with this value) up to an aggregate of scalar and vector types (in this case the total number of elements must match the size of the vector)
- vec ()=default

  Use classical constructors too.

Private Types

- using basic_type = typename detail::small_array< DataType, vec< DataType, NumElements >, Num↩ Elements >

Static Private Member Functions

- template<typename V , typename Element , size_t s>
  static auto flatten (const vec< Element, s > i)

  Flattening helper that does not change scalar values but flatten a vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }.
- template<typename V , typename Type >
  static auto flatten (const Type i)

  If we do not have a vector, just forward it as a tuple up to the final initialization.
- template<typename V , typename... Types>
  static auto flatten_to_tuple (const Types... i)

  Take some initializer values and apply flattening on each value.

Additional Inherited Members

8.11.2.1.1 Member Typedef Documentation

8.11.2.1.1.1 basic_type

```
template<typename DataType, size_t NumElements>
using cl::sycl::vec< DataType, NumElements >::basic_type = typename detail::small_array<Data↩
Type, vec<DataType, NumElements>, NumElements>  [private]
```

Definition at line 47 of file vec.hpp.

8.11.2.1.2 Constructor & Destructor Documentation

8.11.2.1.2.1 vec() [1/2]

```
template<typename DataType, size_t NumElements>
template<typename...  Types>
cl::sycl::vec< DataType, NumElements >::vec (
            const Types...  args )  [inline]
```

Construct a vec from anything from a scalar (to initialize all the elements with this value) up to an aggregate of scalar and vector types (in this case the total number of elements must match the size of the vector)

Definition at line 57 of file vec.hpp.

References cl::sycl::vec< DataType, NumElements >::vec().

```
00058     : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }
```

Here is the call graph for this function:

8.11.2.1.2.2 vec() [2/2]

```
template<typename DataType, size_t NumElements>
cl::sycl::vec< DataType, NumElements >::vec ( )  [default]
```

Use classical constructors too.

Referenced by cl::sycl::vec< DataType, NumElements >::vec().

Here is the caller graph for this function:

8.11.2.1.3 Member Function Documentation

8.11.2.1.3.1 flatten() [1/2]

```
template<typename DataType, size_t NumElements>
template<typename V , typename Element , size_t s>
static auto cl::sycl::vec< DataType, NumElements >::flatten (
            const vec< Element, s > i )  [inline], [static], [private]
```

Flattening helper that does not change scalar values but flatten a vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }.

If we have a vector, just forward its array content since an array has also a tuple interface :-) (23.3.2.9 Tuple interface to class template array [array.tuple])

Definition at line 78 of file vec.hpp.

References i.

```
00078                                                {
00079     static_assert(s <= V::dimension,
00080                   "The element i will not fit in the vector");
00081     return static_cast<std::array<Element, s>>(i);
00082   }
```

8.11.2.1.3.2 flatten() [2/2]

```
template<typename DataType, size_t NumElements>
template<typename V , typename Type >
static auto cl::sycl::vec< DataType, NumElements >::flatten (
            const Type i )  [inline], [static], [private]
```

If we do not have a vector, just forward it as a tuple up to the final initialization.

Returns

typically tuple<double>{ 2.4 } from 2.4 input for example

Definition at line 91 of file vec.hpp.

```
00091                                   {
00092     return std::make_tuple(i);
00093   }
```

8.11.2.1.3.3 flatten_to_tuple()

```
template<typename DataType, size_t NumElements>
template<typename V , typename...  Types>
static auto cl::sycl::vec< DataType, NumElements >::flatten_to_tuple (
            const Types...  i )  [inline], [static], [private]
```

Take some initializer values and apply flattening on each value.

Returns

a tuple of scalar initializer values

Definition at line 101 of file vec.hpp.

References i.

```
00101                                                   {
00102     // Concatenate the tuples returned by each flattening
00103     return std::tuple_cat(flatten<V>(i)...);
00104   }
```

8.11.3 Macro Definition Documentation

8.11.3.1 TRISYCL_DEFINE_VEC_TYPE

```
#define TRISYCL_DEFINE_VEC_TYPE(
              type,
              actual_type )
```

```
#include <include/CL/sycl/vec.hpp>
```

Value:

```
TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type)          \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type)          \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type)          \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type)          \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type)          \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)
```

Declare the vector types of a type for all the sizes.

Definition at line 163 of file vec.hpp.

8.11.3.2 TRISYCL_DEFINE_VEC_TYPE_SIZE

```
#define TRISYCL_DEFINE_VEC_TYPE_SIZE(
              type,
              size,
              actual_type ) using type##size = vec<actual_type, size>;
```

```
#include <include/CL/sycl/vec.hpp>
```

A macro to define type alias, such as for type=uchar, size=4 and actual_type=unsigned char, uchar4 is equivalent to vec<unsigned char, 4>

Definition at line 159 of file vec.hpp.

8.12 Xilinx vendor extensions

Collaboration diagram for Xilinx vendor extensions:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::xilinx
- cl::sycl::xilinx::partition

  Kind of array partition.

Classes

- struct cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >

  Define an array class with partition feature. More...

Macros

- #define _ssdm_op_SpecDataflowPipeline(...) do { } while (0)
- #define _ssdm_op_SpecPipeline(...) do { } while (0)
- #define _ssdm_SpecArrayPartition(...) do { } while (0)

8.12.1 Detailed Description

8.12.2 Class Documentation

8.12.2.1 struct cl::sycl::xilinx::partition_array

template<typename ValueType, std::size_t Size, typename PartitionType = partition::none>
struct cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >

Define an array class with partition feature.

Since on FPGA, users can customize the memory architecture in the system and within the CU. Array partition help us to partition single array to multiple memories that can be accessed simultaneously getting higher memory bandwidth.

Parameters

ValueType	is the type of element.
Size	is the size of the array.
PartitionType	is the array partition type: cyclic, block, and complete. The default type is none.

Todo Deal with multi-dimension array.

Definition at line 156 of file partition_array.hpp.

Collaboration diagram for cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >:

Public Types

- using element_type = ValueType

  Type of array elements.

Public Member Functions

- auto begin ()

  Provide iterator.
- const auto begin () const
- auto end ()
- const auto end () const
- constexpr auto size () const noexcept

  Evaluate size.
- partition_array ()

  Construct an array.
- template<typename SomeContainer >
  partition_array (const SomeContainer &src)

  A constructor from some container.

- template<typename SourceBasicType , typename = std::enable_if_t<std::is_convertible<SourceBasicType, ValueType>::value>> constexpr partition_array (std::initializer_list< SourceBasicType > l)

  Construct an array from initializer_list.
- constexpr ValueType & operator[ ] (std::size_t i)

  Provide a subscript operator.
- constexpr const ValueType & operator[ ] (std::size_t i) const
- constexpr auto get_partition_type () const

  Return the partition type of the array.

Public Attributes

- ValueType elems [Size]

  Store the array elements.

Static Public Attributes

- static constexpr auto array_size = Size

  The number of elements of the 1-D array.
- static constexpr auto partition_type = PartitionType::partition_type

  The kind of partitioning.

8.12.2.1.1 Member Typedef Documentation

8.12.2.1.1.1 element_type

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
using cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::element_type =
ValueType
```

Type of array elements.

Definition at line 168 of file partition_array.hpp.

8.12.2.1.2 Constructor & Destructor Documentation

8.12.2.1.2.1 partition_array() [1/3]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array ( )
[inline]
```

Construct an array.

Definition at line 185 of file partition_array.hpp.

References _ssdm_SpecArrayPartition, cl::sycl::xilinx::partition::block, cl::sycl::xilinx::partition::complete, and cl::sycl::xilinx::partition::cyclic.

```
00185                    {
00186     // Add the intrinsic according expressing to the target compiler the
00187     // partitioning to use
00188     if constexpr (partition_type == partition::type::cyclic)
00189       _ssdm_SpecArrayPartition(&(*this)[0], PartitionType::partition_dim,
00190                                "CYCLIC", PartitionType::physical_mem_num, "");
00191     if constexpr (partition_type == partition::type::block)
00192       _ssdm_SpecArrayPartition(&(*this)[0], PartitionType::partition_dim,
00193                                "BLOCK", PartitionType::ele_in_each_physical_mem,
00194                                "");
00195     if constexpr (partition_type == partition::type::complete)
00196       _ssdm_SpecArrayPartition(&(*this)[0], PartitionType::partition_dim,
00197                                "COMPLETE", 0, "");
00198   }
```

8.12.2.1.2.2 partition_array() [2/3]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
template<typename SomeContainer >
cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array (
            const SomeContainer & src )  [inline]
```

A constructor from some container.

Todo Find a way to specialize this with a safer implementation when the size of src is at least constexpr

Definition at line 203 of file partition_array.hpp.

References src.

```
00204     : partition_array { } {
00205     /// \todo Find a way to specialize this with a safer
00206     /// implementation when the size of src is at least constexpr
00207     std::copy_n(std::begin(src), Size, begin());
00208   }
```

8.12.2.1.2.3 partition_array() [3/3]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
template<typename SourceBasicType , typename = std::enable_if_t<std::is_convertible<Source↩
BasicType, ValueType>::value>>
constexpr cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array
(
            std::initializer_list< SourceBasicType > l )  [inline]
```

Construct an array from initializer_list.

Todo Find a way to specialize this with a safer implementation when the size of src is at least constexpr This does not work... static_assert(l.size() == Size);

Definition at line 218 of file partition_array.hpp.

```
00219     : partition_array { } {
00220     /// \todo Find a way to specialize this with a safer
00221     /// implementation when the size of src is at least constexpr
00222     /// This does not work...
00223     /// static_assert(l.size() == Size);
00224     std::copy_n(std::begin(l), Size, begin());
00225   }
```

8.12.2.1.3 Member Function Documentation

8.12.2.1.3.1 begin() [1/2]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::begin ( )  [inline]
```

Provide iterator.

Definition at line 172 of file partition_array.hpp.

```
00172 { return elems; }
```

8.12.2.1.3.2 begin() [2/2]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
const auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::begin ( )
const  [inline]
```

Definition at line 173 of file partition_array.hpp.

```
00173 { return elems; }
```

8.12.2.1.3.3 end() [1/2]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::end ( )  [inline]
```

Definition at line 174 of file partition_array.hpp.

```
00174 { return elems+Size; }
```

8.12.2.1.3.4 end() [2/2]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
const auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::end ( ) const
[inline]
```

Definition at line 175 of file partition_array.hpp.

```
00175 { return elems+Size; }
```

8.12.2.1.3.5 get_partition_type()

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
constexpr auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::get_↩
partition_type ( ) const  [inline]
```

Return the partition type of the array.

Definition at line 240 of file partition_array.hpp.

```
00240                                                {
00241     return partition_type;
00242   }
```

8.12.2.1.3.6 operator[]() [1/2]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
constexpr ValueType& cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >↩
::operator[] (
            std::size_t i )  [inline]
```

Provide a subscript operator.

Definition at line 229 of file partition_array.hpp.

References i.

```
00229                                                    {
00230     return elems[i];
00231   }
```

8.12.2.1.3.7 operator[]() [2/2]

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
constexpr const ValueType& cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType
>::operator[] (
            std::size_t i ) const  [inline]
```

Definition at line 234 of file partition_array.hpp.

References i.

```
00234                                                          {
00235     return elems[i];
00236   }
```

8.12.2.1.3.8 size()

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
constexpr auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::size ( )
const  [inline], [noexcept]
```

Evaluate size.

Definition at line 179 of file partition_array.hpp.

```
00179                                       {
00180     return Size;
00181   }
```

8.12.2.1.4 Member Data Documentation

8.12.2.1.4.1 array_size

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
constexpr auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::array_size
= Size  [static]
```

The number of elements of the 1-D array.

Definition at line 164 of file partition_array.hpp.

8.12.2.1.4.2 elems

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
ValueType cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::elems[Size]
```

Store the array elements.

Note that it means default initialization for partition_array elements, which is lazily convenient for heterogeneous computing

Definition at line 162 of file partition_array.hpp.

8.12.2.1.4.3 partition_type

```
template<typename ValueType , std::size_t Size, typename PartitionType = partition::none>
constexpr auto cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition↩
_type = PartitionType::partition_type  [static]
```

The kind of partitioning.

Definition at line 166 of file partition_array.hpp.

8.12.3 Macro Definition Documentation

8.12.3.1 _ssdm_op_SpecDataflowPipeline

```
#define _ssdm_op_SpecDataflowPipeline(
              ...  ) do { } while (0)
```

```
#include <include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp>
```

Definition at line 30 of file ssdm_inst.hpp.

Referenced by cl::sycl::xilinx::dataflow().

8.12.3.2 _ssdm_op_SpecPipeline

```
#define _ssdm_op_SpecPipeline(
              ...  ) do { } while (0)
```

```
#include <include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp>
```

Definition at line 31 of file ssdm_inst.hpp.

Referenced by cl::sycl::xilinx::pipeline().

8.12.3.3 _ssdm_SpecArrayPartition

```
#define _ssdm_SpecArrayPartition(
              ...  ) do { } while (0)
```

```
#include <include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp>
```

Definition at line 32 of file ssdm_inst.hpp.

Referenced by cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array().

Chapter 9

Namespace Documentation

9.1 cl Namespace Reference

The vector type to be used as SYCL vector.

Namespaces

- sycl

9.1.1 Detailed Description

The vector type to be used as SYCL vector.

The exception pointer type to be used as SYCL exception pointer.

The hash type to be used as SYCL hash.

The weak pointer type to be used as SYCL weak pointer.

The shared pointer type to be used as SYCL shared pointer.

The unique pointer type to be used as SYCL unique pointer.

The mutex type to be used as SYCL mutex.

The functional type to be used as SYCL function.

The string type to be used as SYCL string.

9.2 cl::sycl Namespace Reference

Namespaces

- access

  Describe the type of access by kernels.
- detail
- drt

  The device-side runtime.
- info
- trisycl
- xilinx

Classes

- class accessor

  The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way. More...
- class accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class accessor< DataType, 1, AccessMode, access::target::pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class accessor_error

  Error regarding the cl::sycl::accessor objects defined. More...
- struct async_exception

  An error stored in an exception_list for asynchronous errors. More...
- class buffer

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- class cl_exception

  Returns the OpenCL error code encapsulated in the exception. More...
- class compile_program_error

  Error while compiling the SYCL kernel to a SYCL device. More...
- class context

  SYCL context. More...
- class device

  SYCL device. More...
- class device_error

  The SYCL device will trigger this exception on error. More...
- class device_selector

  The SYCL heuristics to select a device. More...
- class device_type_selector

  A device selector by device_type. More...
- class device_typename_selector

  Select a device by template device_type parameter. More...
- struct error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...
- class event
- class event_error

Error regarding associated cl::sycl::event objects. More...

- class exception

  Encapsulate a SYCL error information. More...
- struct exception_list

  Exception list to store several exceptions. More...
- class feature_not_supported

  Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on. More...
- struct group

  A group index used in a parallel_for_workitem to specify a work_group. More...
- class handler

  Command group handler class. More...
- class id

  Define a multi-dimensional index, used for example to locate a work item. More...
- struct image
- class invalid_object_error

  Error regarding any memory objects being used inside the kernel. More...
- class invalid_parameter_error

  Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda. More...
- struct is_wrapper
- class item

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- class kernel

  SYCL kernel. More...
- class kernel_error

  Error that occurred before or while enqueuing the SYCL kernel. More...
- class link_program_error

  Error while linking the SYCL kernel to a SYCL device. More...
- class memory_allocation_error

  Error on memory allocation on the SYCL device for a SYCL kernel. More...
- struct nd_item

  A SYCL nd_item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...
- struct nd_range

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...
- class nd_range_error

  Error regarding the cl::sycl::nd_range specified for the SYCL kernel. More...
- class non_cl_error

  Exception for an OpenCL operation requested in a non OpenCL area. More...
- class pipe

  A SYCL pipe. More...
- class pipe_error

  A failing pipe error will trigger this exception on error. More...
- struct pipe_reservation

  The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example. More...
- class platform

  Abstract the OpenCL platform. More...
- class platform_error

  The SYCL platform will trigger this exception on error. More...
- class profiling_error

The SYCL runtime will trigger this error if there is an error when profiling info is enabled. More...

- class queue

  SYCL queue, similar to the OpenCL queue concept. More...
- class range

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...
- class runtime_error
- class static_pipe

  A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. More...
- class vec

  Small OpenCL vector class. More...

Typedefs

- template<typename T >
  using constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using constant_ptr = constant< T * >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using generic = detail::addr_space< T, generic_address_space >

  Declare a variable to be in the OpenCL 2 generic address space.
- template<typename T >
  using global = detail::addr_space< T, global_address_space >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using global_ptr = global< T * >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using local = detail::addr_space< T, local_address_space >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using local_ptr = local< T * >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using priv = detail::addr_space< T, private_address_space >

  Declare a variable to be in the OpenCL private address space.
- template<typename T >
  using private_ptr = priv< T * >

  Declare a variable to be in the OpenCL private address space.
- template<typename Pointer , address_space AS>
  using multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.
- template<typename T >
  using buffer_allocator = std::allocator< T >

  The allocator objects give the programmer some control on how the memory is allocated inside SYCL.
- template<typename T >
  using image_allocator = std::allocator< T >

  The allocator used for the `image` inside SYCL.
- template<typename T >
  using map_allocator = std::allocator< T >

The allocator used to map the memory at the same place.

- template<class T , class Alloc = std::allocator<T>>
  using vector_class = std::vector< T, Alloc >
- using string_class = std::string
- template<class R , class... ArgTypes>
  using function_class = std::function< R(ArgTypes...)>
- using mutex_class = std::mutex
- template<class T , class D = std::default_delete<T>>
  using unique_ptr_class = std::unique_ptr< T[ ], D >
- template<class T >
  using shared_ptr_class = std::shared_ptr< T >
- template<class T >
  using weak_ptr_class = std::weak_ptr< T >
- template<class T >
  using hash_class = std::hash< T >
- using exception_ptr_class = std::exception_ptr
- using default_selector = device_typename_selector< info::device_type::automatic >

  Devices selected by heuristics of the system.
- using gpu_selector = device_typename_selector< info::device_type::gpu >

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.
- using cpu_selector = device_typename_selector< info::device_type::cpu >

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.
- using host_selector = device_typename_selector< info::device_type::host >

  Selects the SYCL host CPU device that does not require an OpenCL runtime.
- using exception_ptr = std::exception_ptr

  A shared pointer to an exception as in C++ specification.
- using async_handler = function_class< void, exception_list >

Enumerations

- enum address_space {
  constant_address_space, generic_address_space, global_address_space, local_address_space, private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename Accessor >
  static auto & get_pipe_detail (Accessor &a)

  Top-level function to break circular dependencies on the the types to get the pipe implementation.
- template<typename T , address_space AS>
  multi_ptr< T, AS > make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.
- template<>
  auto device::get_info< info::device::max_work_group_size > () const
- template<>
  auto device::get_info< info::device::max_compute_units > () const
- template<>
  auto device::get_info< info::device::device_type > () const
- template<>
  auto device::get_info< info::device::local_mem_size > () const

- template<>
  auto device::get_info< info::device::max_mem_alloc_size > () const
- template<>
  auto device::get_info< info::device::vendor > () const
- template<>
  auto device::get_info< info::device::name > () const
- template<>
  auto device::get_info< info::device::profile > () const
- auto make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto make_id (id< 2 > i)
- auto make_id (id< 3 > i)
- template<typename... BasicType>
  auto make_id (BasicType... Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)
- TRISYCL_MATH_WRAP (abs) TRISYCL_MATH_WRAP(atan) TRISYCL_MATH_WRAP2s(fmax) TRISYCL_MATH_WRAP2s(fmin) TRISYCL_MATH_WRAP2s(frexp) template< typename T > T max(T x
- template<typename T >
  T min (T x, T y, T z)
- TRISYCL_MATH_WRAP2s (modf) TRISYCL_MATH_WRAP3s(remquo) TRISYCL_MATH_WRAP2(rotate)
  namespace native
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.
- auto make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto make_range (range< 2 > r)
- auto make_range (range< 3 > r)
- template<typename... BasicType>
  auto make_range (BasicType... Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

Variables

- T y
- T T z

9.2.1 Typedef Documentation

9.2.1.1 exception_ptr_class

```
using cl::sycl::exception_ptr_class = typedef std::exception_ptr
```

Definition at line 143 of file default_classes.hpp.

9.2.1.2 function_class

```
template<class R , class...  ArgTypes>
using cl::sycl::function_class = typedef std::function<R(ArgTypes...)>
```

Definition at line 55 of file default_classes.hpp.

9.2.1.3 hash_class

```
template<class T >
using cl::sycl::hash_class = typedef std::hash<T>
```

Definition at line 129 of file default_classes.hpp.

9.2.1.4 mutex_class

```
using cl::sycl::mutex_class = typedef std::mutex
```

Definition at line 69 of file default_classes.hpp.

9.2.1.5 shared_ptr_class

```
template<class T >
using cl::sycl::shared_ptr_class = typedef std::shared_ptr<T>
```

Definition at line 99 of file default_classes.hpp.

9.2.1.6 string_class

```
using cl::sycl::string_class = typedef std::string
```

Definition at line 40 of file default_classes.hpp.

9.2.1.7 unique_ptr_class

```
template<class T , class D = std::default_delete<T>>
using cl::sycl::unique_ptr_class = typedef std::unique_ptr<T[], D>
```

Definition at line 84 of file default_classes.hpp.

9.2.1.8 vector_class

```
template<class T , class Alloc = std::allocator<T>>
using cl::sycl::vector_class = typedef std::vector<T, Alloc>
```

Definition at line 26 of file default_classes.hpp.

9.2.1.9 weak_ptr_class

```
template<class T >
using cl::sycl::weak_ptr_class = typedef std::weak_ptr<T>
```

Definition at line 114 of file default_classes.hpp.

9.2.2 Function Documentation

9.2.2.1 min()

```
template<typename T >
T cl::sycl::min (
            T x,
            T y,
            T z )
```

Definition at line 123 of file math.hpp.

```
00123                       {
00124   return std::min(x, std::min(y, z));
00125 }
```

9.2.2.2 TRISYCL_MATH_WRAP()

```
cl::sycl::TRISYCL_MATH_WRAP (
            abs  )
```

Referenced by TRISYCL_MATH_WRAP2s().

Here is the caller graph for this function:

9.2.2.3 TRISYCL_MATH_WRAP2s()

```
cl::sycl::TRISYCL_MATH_WRAP2s (
            modf  )
```

Definition at line 133 of file math.hpp.

References exp, and TRISYCL_MATH_WRAP().

```
00171                 {
00172 TRISYCL_MATH_WRAP(cos)
00173 //*TRISYCL_MATH_WRAP2(divide)
00174 TRISYCL_MATH_WRAP(exp)
00175 TRISYCL_MATH_WRAP(exp2)
00176 //*TRISYCL_MATH_WRAP(exp10)
00177 TRISYCL_MATH_WRAP(log)
00178 TRISYCL_MATH_WRAP(log2)
00179 TRISYCL_MATH_WRAP(log10)
00180 //*TRISYCL_MATH_WRAP(powr)
00181 //*TRISYCL_MATH_WRAP(recip)
00182 //*TRISYCL_MATH_WRAP(rsqrt)
00183 TRISYCL_MATH_WRAP(sin)
00184 TRISYCL_MATH_WRAP(sqrt)
00185 TRISYCL_MATH_WRAP(tan)
00186 }
```

Here is the call graph for this function:

9.2.3 Variable Documentation

9.2.3.1 y

```
T cl::sycl::y
```

Definition at line 110 of file math.hpp.

9.2.3.2 z

```
T T cl::sycl::z
```

Initial value:

```
{
  return std::max(x, std::max(y, z))
```

Definition at line 110 of file math.hpp.

9.3 cl::sycl::access Namespace Reference

Describe the type of access by kernels.

Enumerations

- enum mode {
  mode::read = 42, mode::write, mode::read_write, mode::discard_write,
  mode::discard_read_write, mode::atomic }

  This describes the type of the access mode to be used via accessor.
- enum target {
  target::global_buffer = 2014, target::constant_buffer, target::local, target::image,
  target::host_buffer, target::host_image, target::image_array, target::pipe,
  target::blocking_pipe }

  The target enumeration describes the type of object to be accessed via the accessor.
- enum fence_space : char { fence_space::local_space, fence_space::global_space, fence_space::global_↩
  and_local }

  Precise the address space a barrier needs to act on.

9.3.1 Detailed Description

Describe the type of access by kernels.

Todo This values should be normalized to allow separate compilation with different implementations?

9.3.2 Enumeration Type Documentation

9.3.2.1 fence_space

`enum cl::sycl::access::fence_space : char [strong]`

Precise the address space a barrier needs to act on.

Enumerator

local_space	
global_space	
global_and_local	

Definition at line 63 of file access.hpp.

```
00063                          : char {
00064     local_space,
00065     global_space,
00066     global_and_local
00067   };
```

9.3.2.2 mode

```
enum cl::sycl::access::mode [strong]
```

This describes the type of the access mode to be used via accessor.

Enumerator

read	Read-only access. Insist on the fact that read_write != read + write
write	Write-only access, but previous content *not* discarded.
read_write	Read and write access.
discard_write	Write-only access and previous content discarded.
discard_read_write	Read and write access and previous content discarded.
atomic	Atomic access.

Definition at line 33 of file access.hpp.

```
00033                    {
00034      read = 42, /**< Read-only access. Insist on the fact that
00035                      read_write != read + write */
00036      write, ///< Write-only access, but previous content *not* discarded
00037      read_write, ///< Read and write access
00038      discard_write, ///< Write-only access and previous content discarded
00039      discard_read_write, /**< Read and write access and previous
00040                                content discarded*/
00041      atomic ///< Atomic access
00042    };
```

9.3.2.3 target

```
enum cl::sycl::access::target [strong]
```

The target enumeration describes the type of object to be accessed via the accessor.

Enumerator

global_buffer	
constant_buffer	
local	
image	
host_buffer	
host_image	
image_array	
pipe	
blocking_pipe	

Definition at line 48 of file access.hpp.

```
                    {
    global_buffer = 2014, //< Just pick a random number...
    constant_buffer,
    local,
    image,
    host_buffer,
    host_image,
    image_array,
    pipe,
    blocking_pipe
  };
```

9.4 cl::sycl::detail Namespace Reference

Classes

- class accessor

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- class accessor< T, Dimensions, Mode, access::target::local >

  The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group. More...
- class accessor_base

  The buffer accessor base is here mainly to be accessed from the task without dependency on the buffer data types. More...
- struct address_space_array

  Implementation of an array variable with an OpenCL address space. More...
- struct address_space_base

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct address_space_fundamental

  Implementation of a fundamental type with an OpenCL address space. More...
- struct address_space_object

  Implementation of an object type with an OpenCL address space. More...
- struct address_space_ptr

  Implementation for an OpenCL address space pointer. More...
- struct address_space_variable

  Implementation of a variable with an OpenCL address space. More...
- class buffer

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- struct buffer_base

  Factorize some template independent buffer aspects in a base class. More...
- class buffer_waiter

  A helper class to wait for the final buffer destruction if the conditions for blocking are met. More...
- class cache

  A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.
- struct container_element_aspect

  A mix-in to add some container element aspects. More...
- class context
- struct debug

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- class device

  An abstract class representing various models of SYCL devices. More...

- struct display_vector

  Class used to display a vector-like type of classes that inherit from it. More...
- struct expand_to_vector

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...
- class host_context
- class host_device

  SYCL host device.
- class host_platform

  SYCL host platform. More...
- class host_queue

  Some implementation details about the SYCL queue.
- class kernel

  Abstract SYCL kernel. More...
- struct ocl_type

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct ocl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct ocl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct ocl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct ocl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- class opencl_context

  SYCL OpenCL context.
- class opencl_device

  SYCL OpenCL device.
- class opencl_kernel

  An abstraction of the OpenCL kernel.
- class opencl_platform

  SYCL OpenCL platform. More...
- class opencl_queue

  Some implementation details about the SYCL queue.
- struct parallel_for_iterate

  A recursive multi-dimensional iterator that ends up calling f. More...
- struct parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...
- struct parallel_OpenMP_for_iterate

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- class pipe

  Implement a pipe object. More...
- class pipe_accessor

  The accessor abstracts the way pipe data are accessed inside a kernel. More...
- class pipe_reservation

  The implementation of the pipe reservation station. More...
- class platform

  An abstract class representing various models of SYCL platforms. More...
- struct queue

Some implementation details about the SYCL queue.

- struct reserve_id

  A private description of a reservation station. More...

- struct shared_ptr_implementation

  Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

- struct singleton

  Provide a singleton factory.

- struct small_array

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...

- struct small_array_123

  A small array of 1, 2 or 3 elements with the implicit constructors. More...

- struct small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if Dimensions = 1. More...

- struct small_array_123< BasicType, FinalType, 2 >
- struct small_array_123< BasicType, FinalType, 3 >
- struct task

  The abstraction to represent SYCL tasks executing inside command_group.

Typedefs

- template<typename T , address_space AS>
  using addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std↩::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.

Functions

- template<typename BufferDetail >
  static std::shared_ptr< detail::task > buffer_add_to_task (BufferDetail buf, handler *command_group_↩handler, bool is_write_mode)

  Proxy function to avoid some circular type recursion.

- static std::shared_ptr< detail::task > add_buffer_to_task (handler *command_group_handler, std::shared↩_ptr< detail::buffer_base > b, bool is_write_mode)
- template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>
  auto waiter (detail::buffer< T, Dimensions > *b)

  Helper function to create a new buffer_waiter.

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > tuple_to_array_iterate (Tuple t, std::index_↩sequence< Is... >)

  Helper to construct an array from initializer elements provided as a tuple.

- template<typename V , typename Tuple >
  auto tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.

- template<typename V , typename Tuple >
  auto expand (Tuple t)

  Create the array data of V from a tuple of initializer.

- template<typename KernelName , typename Functor >
  auto trace_kernel (Functor f)

Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

- template<typename KernelName , typename Functor >
  __attribute__ ((noinline)) void instantiate_kernel(Functor f) noexcept

  Instantiate the kernel code.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX set_kernel_task_marker (detail::task &) noexcept

  Set the task to be used to launch the kernel.
- template<typename Range , typename Id >
  size_t constexpr linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.
- void unimplemented ()

  Display an "unimplemented" message.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.
- static std::shared_ptr< detail::task > add_buffer_to_task (handler *command_group_handler, std::shared↩_ptr< detail::buffer_base > b, bool is_write_mode)

  Register a buffer as used by a task.
- template<int Dimensions = 1, typename ParallelForFunctor , typename Id >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f, Id)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f, item< Dimensions >)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelFor↩Functor f)

  Implementation of parallel_for with a range<> and an offset.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<int Dimensions = 1, typename ParallelForFunctor >
  void parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.

Variables

- TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_context, detail::opencl_context > opencl_context↩::cache TRISYCL_WEAK_ATTRIB_SUFFIX
- auto prevent_arguments_from_optimization

  Avoid the interprocedural optimization to remove these arguments in the kernel instantiation by relying on some dummy static variables.

9.4.1 Function Documentation

9.4.1.1 __attribute__()

```
template<typename KernelName , typename Functor >
cl::sycl::detail::__attribute__ (
            (noinline)  )  [noexcept]
```

Instantiate the kernel code.

To have a clear view of what a kernel is at the LLVM IR level even when a lot of optimizations are done (-O3), call the kernel from this function marked as `noinline`, otherwise the kernel code is inlined directly where it is used.

Actually some optimizations/code restructuring might duplicate the call to `instantiate_kernel` but the `noduplicate` attribute seems incompatible with `noinline` in Clang/LLVM 3.9.1, because the inlining happen when both attributes are used...

Definition at line 52 of file instantiate_kernel.hpp.

References TRISYCL_WEAK_ATTRIB_PREFIX, and TRISYCL_WEAK_ATTRIB_SUFFIX.

Referenced by set_kernel_task_marker().

```
00053                                       {
00054   /* The outlining compiler is expected to do some massage here or
00055      around and to insert some calls to \c serialize_arg and so on */
00056   f();
00057 }
```

Here is the caller graph for this function:

9.4.1.2 add_buffer_to_task()

```
static std::shared_ptr<detail::task> cl::sycl::detail::add_buffer_to_task (
            handler * command_group_handler,
            std::shared_ptr< detail::buffer_base > b,
            bool is_write_mode )  [static]
```

Register a buffer as used by a task.

This is a proxy function to avoid complicated type recursion.

Definition at line 468 of file handler.hpp.

References cl::sycl::handler::task.

```
00470                                       {
00471   command_group_handler->task->add_buffer(b, is_write_mode);
00472   return command_group_handler->task;
00473 }
```

9.4.1.3 set_kernel_task_marker()

```
TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::detail::set_kernel_task↩
_marker (
            detail::task &  )  [noexcept]
```

Set the task to be used to launch the kernel.

This is actually a dummy function so that the device compiler can just grab the task for passing it to the argument serialization functions from the device runtime.

It is better to use a weak linkage than having this function with extern linkage:

- if the device compiler is used the function is kept by the compiler because full link and LTO is not applied before the device compiler passes are applied. So there is no partial evaluation in the compiler according to what is returned by this function;
- if the device compiler is not used, the code still links and can be compiled.

Definition at line 79 of file instantiate_kernel.hpp.

References __attribute__().

```
00079                                                   {
00080 }
```

Here is the call graph for this function:

9.4.2 Variable Documentation

9.4.2.1 prevent_arguments_from_optimization

```
auto cl::sycl::detail::prevent_arguments_from_optimization
```

Initial value:

```
= [] (auto & ...args) {

  static auto __attribute__((used)) keep = std::make_tuple(& args...);
}
```

Avoid the interprocedural optimization to remove these arguments in the kernel instantiation by relying on some dummy static variables.

Using **attribute**((used)) does not work on arguments but only on static variable, so use this function.

Definition at line 30 of file instantiate_kernel.hpp.

9.5 cl::sycl::drt Namespace Reference

The device-side runtime.

Namespaces

- code

  The binary code of the kernels.

Classes

- class accessor

  SYCL accessor seen from a device perspective. More...

Functions

- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX serialize_arg (detail::task &task, std::size_t index, void ∗arg, std::size_t arg_size)

  Set an argument value of the kernel.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX serialize_accessor_arg (detail::task &task, std::size_t index, void ∗arg, std::size_t arg_size)

  Set an accessor argument of the kernel.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX set_kernel (detail::task &task, const char ∗kernel_name, const char ∗kernel_short_name)

  Set the kernel for a later invocation.

Variables

- TRISYCL_WEAK_ATTRIB_PREFIX std::map< std::string, std::vector< unsigned char > > TRISYCL_W↩EAK_ATTRIB_SUFFIX kernel_IR

  The code of the kernels arranged by name.

9.5.1 Detailed Description

The device-side runtime.

9.6 cl::sycl::drt::code Namespace Reference

The binary code of the kernels.

Classes

- struct program

  A binary program containing some kernels.

9.6.1 Detailed Description

The binary code of the kernels.

9.7 cl::sycl::info Namespace Reference

Classes

- struct param_traits

  Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Typedefs

- using device_fp_config = unsigned int
- using device_exec_capabilities = unsigned int
- using device_queue_properties = unsigned int
- using queue_profiling = bool

Enumerations

- enum context : int { context::reference_count, context::platform, context::devices }

  Context information descriptors.
- enum device_type : unsigned int {
  device_type::cpu, device_type::gpu, device_type::accelerator, device_type::custom,
  device_type::automatic, device_type::host, device_type::opencl, device_type::all }

  Type of devices.
- enum device : int {
  device::device_type, device::vendor_id, device::max_compute_units, device::max_work_item_dimensions,
  device::max_work_item_sizes, device::max_work_group_size, device::preferred_vector_width_char,
  device::preferred_vector_width_short,
  device::preferred_vector_width_int, device::preferred_vector_width_long_long, device::preferred_vector_width_float, device::preferred_vector_width_double,
  device::preferred_vector_width_half, device::native_vector_witdth_char, device::native_vector_witdth_short,
  device::native_vector_witdth_int,
  device::native_vector_witdth_long_long, device::native_vector_witdth_float, device::native_vector_witdth_double, device::native_vector_witdth_half,
  device::max_clock_frequency, device::address_bits, device::max_mem_alloc_size, device::image_support,
  device::max_read_image_args, device::max_write_image_args, device::image2d_max_height, device::image2d_max_width,
  device::image3d_max_height, device::image3d_max_widht, device::image3d_mas_depth, device::image_max_buffer_size,
  device::image_max_array_size, device::max_samplers, device::max_parameter_size, device::mem_base_addr_align,
  device::single_fp_config, device::double_fp_config, device::global_mem_cache_type, device::global_mem_cache_line_size,
  device::global_mem_cache_size, device::global_mem_size, device::max_constant_buffer_size, device::max_constant_args,
  device::local_mem_type, device::local_mem_size, device::error_correction_support, device::host_unified_memory,
  device::profiling_timer_resolution, device::endian_little, device::is_available, device::is_compiler_available,

device::is_linker_available, device::execution_capabilities, device::queue_properties, device::built_in_kernels,
device::platform, device::name, device::vendor, device::driver_version,
device::profile, device::device_version, device::opencl_version, device::extensions,
device::printf_buffer_size, device::preferred_interop_user_sync, device::parent_device, device::partition_max_sub_devices,
device::partition_properties, device::partition_affinity_domain, device::partition_type, device::reference_count }

Device information descriptors.

- enum device_partition_property : int {
device_partition_property::unsupported, device_partition_property::partition_equally, device_partition_property::partition_by_counts, device_partition_property::partition_by_affinity_domain,
device_partition_property::partition_affinity_domain_next_partitionable }
- enum device_affinity_domain : int {
device_affinity_domain::unsupported, device_affinity_domain::numa, device_affinity_domain::L4_cache, device_affinity_domain::L3_cache,
device_affinity_domain::L2_cache, device_affinity_domain::next_partitionable }
- enum device_partition_type : int {
device_partition_type::no_partition, device_partition_type::numa, device_partition_type::L4_cache, device_partition_type::L3_cache,
device_partition_type::L2_cache, device_partition_type::L1_cache }
- enum local_mem_type : int { local_mem_type::none, local_mem_type::local, local_mem_type::global }
- enum fp_config : int {
fp_config::denorm, fp_config::inf_nan, fp_config::round_to_nearest, fp_config::round_to_zero,
fp_config::round_to_inf, fp_config::fma, fp_config::correctly_rounded_divide_sqrt, fp_config::soft_float }
- enum global_mem_cache_type : int { global_mem_cache_type::none, global_mem_cache_type::read_only, global_mem_cache_type::write_only }
- enum device_execution_capabilities : unsigned int { device_execution_capabilities::exec_kernel, device_execution_capabilities::exec_native_kernel }
- enum platform : unsigned int {
platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_PROFILE), platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VERSION), platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_NAME), platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VENDOR),
platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_EXTENSIONS) }

  Platform information descriptors.
- enum queue : int { queue::context, queue::device, queue::reference_count, queue::properties }

  Queue information descriptors.

9.7.1 Typedef Documentation

9.7.1.1 queue_profiling

```
using cl::sycl::info::queue_profiling = typedef bool
```

Definition at line 46 of file queue.hpp.

9.7.2 Enumeration Type Documentation

9.7.2.1 queue

```
enum cl::sycl::info::queue : int [strong]
```

Queue information descriptors.

From specification C.4

Todo unsigned int?

Todo To be implemented

Enumerator

context	
device	
reference_count	
properties	

Definition at line 56 of file queue.hpp.

```
00056                    : int {
00057   context,
00058   device,
00059   reference_count,
00060   properties
00061 };
```

9.8 cl::sycl::trisycl Namespace Reference

Classes

- struct default_error_handler

9.8.1 Detailed Description

Todo Refactor when updating to latest specification

9.9 cl::sycl::xilinx Namespace Reference

Namespaces

- partition

  Kind of array partition.

Classes

- struct partition_array

  Define an array class with partition feature. More...

Functions

- template<typename Functor >
  void dataflow (Functor f) noexcept

  Apply dataflow execution on functions or loops.
- template<typename Functor >
  void pipeline (Functor f) noexcept

  Execute loops in a pipelined manner.

9.9.1 Function Documentation

9.9.1.1 dataflow()

```
template<typename Functor >
void cl::sycl::xilinx::dataflow (
            Functor f )  [noexcept]
```

Apply dataflow execution on functions or loops.

With this mode, Xilinx tools analyze the dataflow dependencies between sequential functions or loops and create channels (based on ping-pong RAMs or FIFOs) that allow consumer functions or loops to start operation before the producer functions or loops have completed.

This allows functions or loops to operate in parallel, which decreases latency and improves the throughput.

Parameters

in	*f*	is a function that functions or loops in f will be executed in a dataflow manner.

Definition at line 34 of file opt_decorate_func.hpp.

References _ssdm_op_SpecDataflowPipeline.

```
{
  /* SSDM instruction is inserted before the argument functor to guide xocc to
     do dataflow. */
  _ssdm_op_SpecDataflowPipeline(-1, "");
  f();
}
```

9.9.1.2 pipeline()

```
template<typename Functor >
void cl::sycl::xilinx::pipeline (
            Functor f )  [noexcept]
```

Execute loops in a pipelined manner.

A loop with pipeline mode processes a new input every clock cycle. This allows the operations of different iterations of the loop to be executed in a concurrent manner to reduce latency.

Parameters

in	*f*	is a function with an innermost loop to be executed in a pipeline way.

Definition at line 52 of file opt_decorate_func.hpp.

References _ssdm_op_SpecPipeline.

```
00052                                   {
00053   /* SSDM instruction is inserted before the argument functor to guide xocc to
00054      do pipeline. */
00055   _ssdm_op_SpecPipeline(1, 1, 0, 0, "");
00056   f();
00057 }
```

9.10 cl::sycl::xilinx::partition Namespace Reference

Kind of array partition.

Classes

- struct block

  Represent a block partition.
- struct complete

  Represent a complete partition.
- struct cyclic

  Represent a cyclic partition.
- struct none

  Represent a none partition.

Enumerations

- enum type { type::cyclic, type::block, type::complete, type::none }

  Three real partition types: cyclic, block, complete.

9.10.1 Detailed Description

Kind of array partition.

To be used when defining or declaring a vendor-supported partition array in kernel.

9.10.2 Enumeration Type Documentation

9.10.2.1 type

```
enum cl::sycl::xilinx::partition::type [strong]
```

Three real partition types: cyclic, block, complete.

none represents non partitioned standard array.

Enumerator

cyclic	
block	
complete	
none	

Definition at line 38 of file partition_array.hpp.

```
00038                    {
00039     cyclic,
00040     block,
00041     complete,
00042     none
00043   };
```

9.11 std Namespace Reference

Classes

- struct hash< cl::sycl::buffer< T, Dimensions, Allocator > >
- struct hash< cl::sycl::context >
- struct hash< cl::sycl::device >
- struct hash< cl::sycl::kernel >
- struct hash< cl::sycl::platform >
- struct hash< cl::sycl::queue >

Chapter 10

Class Documentation

10.1 cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim > Struct Template Reference

Represent a block partition.

```
#include <partition_array.hpp>
```

Collaboration diagram for cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >:

Static Public Attributes

- static constexpr auto ele_in_each_physical_mem = ElmInEachPhyMem
- static constexpr auto partition_dim = PDim
- static constexpr auto partition_type = type::block

10.1.1 Detailed Description

template<std::size_t ElmInEachPhyMem = 1, std::size_t PDim = 1>
struct cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >

Represent a block partition.

The single array would be partitioned into several small physical memories and can be accessed simultaneously, too. However, the first physical memory will be filled up first, then the next.

That is if we have a 4-element array which contains 4 integers 0, 1, 2, and 3. If we set factor to 2, and partition dimension to 1 for this cyclic partition array. Then, the contents of this array will be distributed to 2 physical memories: one contains 1, 2 and the other contains 3,4.

Parameters

ElmInEachPhyMem	is the number of elements in each physical memory that user wants to have.
PDim	is the dimension that user wants to apply block partition on. If PDim is 0, all dimensions will be partitioned with block order.

Todo Deal with multi-dimension array. Now, since we can only deal with 1-dim, PDim is set to 1 by default.

Definition at line 100 of file partition_array.hpp.

10.1.2 Member Data Documentation

10.1.2.1 ele_in_each_physical_mem

```
template<std::size_t ElmInEachPhyMem = 1, std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >::ele_in_each_↩
physical_mem = ElmInEachPhyMem  [static]
```

Definition at line 101 of file partition_array.hpp.

10.1.2.2 partition_dim

```
template<std::size_t ElmInEachPhyMem = 1, std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >::partition_dim =
PDim  [static]
```

Definition at line 102 of file partition_array.hpp.

10.1.2.3 partition_type

```
template<std::size_t ElmInEachPhyMem = 1, std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >::partition_type =
type::block  [static]
```

Definition at line 103 of file partition_array.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/vendor/Xilinx/partition_array.hpp

10.2 cl::sycl::detail::cache< Key, Value > Class Template Reference

A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.

```
#include <cache.hpp>
```

Collaboration diagram for cl::sycl::detail::cache< Key, Value >:

Public Types

- using key_type = Key

  The type of the keys used to indexed the cache.
- using value_type = Value

  The base type of the values stored in the cache.

Public Member Functions

- template<typename Functor >
  std::shared_ptr< value_type > get_or_register (const key_type &k, Functor &&create_element)

  Get a value stored in the cache if present or insert by calling a generator function.
- void remove (const key_type &k)

  Remove an entry from the cache.

Private Attributes

- std::unordered_map< key_type, std::weak_ptr< value_type > > c

  The caching storage.
- std::mutex m

  To make the cache thread-safe.

10.2.1 Detailed Description

template<typename Key, typename Value>
class cl::sycl::detail::cache< Key, Value >

A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.

Since internally only std::weak_ptr are stored, this does not prevent object deletion but it is up to the programmer not to use this cache to retrieve deleted objects.

Definition at line 29 of file cache.hpp.

10.2.2 Member Typedef Documentation

10.2.2.1 key_type

```
template<typename Key, typename Value>
using cl::sycl::detail::cache< Key, Value >::key_type = Key
```

The type of the keys used to indexed the cache.

Definition at line 34 of file cache.hpp.

10.2.2.2 value_type

```
template<typename Key, typename Value>
using cl::sycl::detail::cache< Key, Value >::value_type = Value
```

The base type of the values stored in the cache.

Definition at line 37 of file cache.hpp.

10.2.3 Member Function Documentation

10.2.3.1 get_or_register()

```
template<typename Key, typename Value>
template<typename Functor >
std::shared_ptr<value_type> cl::sycl::detail::cache< Key, Value >::get_or_register (
            const key_type & k,
            Functor && create_element )  [inline]
```

Get a value stored in the cache if present or insert by calling a generator function.

Parameters

in	*k*	is the key used to retrieve the value
in	*create_element*	is the function to be called if the key is not found in the cache to generate a value which is inserted for the key. This function has to produce a value convertible to a shared_ptr

Returns

a shared_ptr to the value retrieved or inserted

Definition at line 62 of file cache.hpp.

Referenced by cl::sycl::detail::opencl_kernel::instance(), cl::sycl::detail::opencl_queue::instance(), cl::sycl::detail::opencl_platform::instance(), cl::sycl::detail::opencl_context::instance(), and cl::sycl::detail::opencl_device::instance().

```
00063                                                                    {
00064     std::lock_guard<std::mutex> lg { m };
00065
00066     auto i = c.find(k);
00067     if (i != c.end())
00068       if (auto observe = i->second.lock())
00069         // Returns \c shared_ptr only if target object is still alive
00070         return observe;
00071
00072     // Otherwise create and insert a new element
00073     std::shared_ptr<value_type> e { create_element() };
00074     c.insert({ k, e });
00075     return e;
00076   }
```

Here is the caller graph for this function:

10.2.3.2 remove()

```
template<typename Key, typename Value>
void cl::sycl::detail::cache< Key, Value >::remove (
            const key_type & k )  [inline]
```

Remove an entry from the cache.

Parameters

in	*k*	is the key associated to the value to remove from the cache

Definition at line 84 of file cache.hpp.

Referenced by cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_RANGE(), cl::sycl::detail::opencl_context::~opencl_context(), cl::sycl::detail::opencl_device::~opencl_device(), cl::sycl::detail::opencl_platform::~opencl_platform(), and cl::sycl::detail::opencl_queue::~opencl_queue().

```
00084                                 {
00085     std::lock_guard<std::mutex> lg { m };
00086     c.erase(k);
00087   }
```

Here is the caller graph for this function:

10.2.4 Member Data Documentation

10.2.4.1 c

```
template<typename Key, typename Value>
std::unordered_map<key_type, std::weak_ptr<value_type> > cl::sycl::detail::cache< Key, Value
>::c  [private]
```

The caching storage.

Definition at line 42 of file cache.hpp.

10.2.4.2 m

```
template<typename Key, typename Value>
std::mutex cl::sycl::detail::cache< Key, Value >::m [private]
```

To make the cache thread-safe.

Definition at line 45 of file cache.hpp.

The documentation for this class was generated from the following file:

- include/CL/sycl/detail/cache.hpp

10.3 cl::sycl::xilinx::partition::complete< PDim > Struct Template Reference

Represent a complete partition.

```
#include <partition_array.hpp>
```

Collaboration diagram for cl::sycl::xilinx::partition::complete< PDim >:

Static Public Attributes

- static constexpr auto partition_dim = PDim
- static constexpr auto partition_type = type::complete

10.3.1 Detailed Description

template<std::size_t PDim = 1>
struct cl::sycl::xilinx::partition::complete< PDim >

Represent a complete partition.

The single array would be partitioned into individual elements. That is if we have a 4-element array with one dimension, the array is completely partitioned into distributed RAM or 4 independent registers.

Parameters

PDim	is the dimension that user wants to apply complete partition on. If PDim is 0, all dimensions will be completely partitioned.

Todo Deal with multi-dimension array. Now, since we can only deal with 1-dim, PDim is set to 1 by default.

Definition at line 121 of file partition_array.hpp.

10.3.2 Member Data Documentation

10.3.2.1 partition_dim

```
template<std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::complete< PDim >::partition_dim = PDim  [static]
```

Definition at line 122 of file partition_array.hpp.

10.3.2.2 partition_type

```
template<std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::complete< PDim >::partition_type = type::complete
[static]
```

Definition at line 123 of file partition_array.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/vendor/Xilinx/partition_array.hpp

10.4 cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim > Struct Template Reference

Represent a cyclic partition.

```
#include <partition_array.hpp>
```

Collaboration diagram for cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >:

Static Public Attributes

- static constexpr auto physical_mem_num = PhyMemNum
- static constexpr auto partition_dim = PDim
- static constexpr auto partition_type = type::cyclic

10.4.1 Detailed Description

template<std::size_t PhyMemNum = 1, std::size_t PDim = 1>
struct cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >

Represent a cyclic partition.

The single array would be partitioned into several small physical memories in this case. These small physical memories can be accessed simultaneously which drive the performance. Each element in the array would be partitioned to each memory in order and cyclically.

That is if we have a 4-element array which contains 4 integers 0, 1, 2, and 3. If we set factor to 2, and partition dimension to 1 for this cyclic partition array. Then, the contents of this array will be distributed to 2 physical memories: one contains 1, 3 and the other contains 2,4.

Parameters

PhyMemNum	is the number of physical memories that user wants to have.
PDim	is the dimension that user wants to apply cyclic partition on. If PDim is 0, all dimensions will be partitioned with cyclic order.

Todo Deal with multi-dimension array. Now, since we can only deal with 1-dim, PDim is set to 1 by default.

Definition at line 70 of file partition_array.hpp.

10.4.2 Member Data Documentation

10.4.2.1 partition_dim

```
template<std::size_t PhyMemNum = 1, std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >::partition_dim = PDim
[static]
```

Definition at line 72 of file partition_array.hpp.

10.4.2.2 partition_type

```
template<std::size_t PhyMemNum = 1, std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >::partition_type = type↩
::cyclic  [static]
```

Definition at line 73 of file partition_array.hpp.

10.4.2.3 physical_mem_num

```
template<std::size_t PhyMemNum = 1, std::size_t PDim = 1>
constexpr auto cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >::physical_mem_num =
PhyMemNum  [static]
```

Definition at line 71 of file partition_array.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/vendor/Xilinx/partition_array.hpp

10.5 cl::sycl::trisycl::default_error_handler Struct Reference

```
#include <error_handler.hpp>
```

Inheritance diagram for cl::sycl::trisycl::default_error_handler:

Collaboration diagram for cl::sycl::trisycl::default_error_handler:

Public Member Functions

- void report_error (exception &) override

  The method to define to be called in the case of an error.

Additional Inherited Members

10.5.1 Detailed Description

Definition at line 52 of file error_handler.hpp.

10.5.2 Member Function Documentation

10.5.2.1 report_error()

```
void cl::sycl::trisycl::default_error_handler::report_error (
            exception & error )  [inline], [override], [virtual]
```

The method to define to be called in the case of an error.

Todo Add "virtual void" to the specification

Implements cl::sycl::error_handler.

Definition at line 54 of file error_handler.hpp.

```
00054                                                   {
00055     }
```

The documentation for this struct was generated from the following file:

- include/CL/sycl/error_handler.hpp

10.6 cl::sycl::event Class Reference

```
#include <event.hpp>
```

Public Member Functions

- event ()=default

10.6.1 Detailed Description

Definition at line 14 of file event.hpp.

10.6.2 Constructor & Destructor Documentation

10.6.2.1 event()

```
cl::sycl::event::event ( ) [default]
```

The documentation for this class was generated from the following file:

- include/CL/sycl/event.hpp

10.7 handler_event Class Reference

Handler event.

```
#include <handler_event.hpp>
```

10.7.1 Detailed Description

Handler event.

Todo To be implemented

Todo To be implemented

Definition at line 19 of file handler_event.hpp.

The documentation for this class was generated from the following file:

- include/CL/sycl/handler_event.hpp

10.8 std::hash< cl::sycl::buffer< T, Dimensions, Allocator > > Struct Template Reference

```
#include <buffer.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::buffer< T, Dimensions, Allocator > &b) const

10.8.1 Detailed Description

template<typename T, int Dimensions, typename Allocator>
struct std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >

Definition at line 542 of file buffer.hpp.

10.8.2 Member Function Documentation

10.8.2.1 operator()()

```
template<typename T , int Dimensions, typename Allocator >
auto std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >::operator() (
            const cl::sycl::buffer< T, Dimensions, Allocator > & b ) const  [inline]
```

Definition at line 544 of file buffer.hpp.

References cl::sycl::detail::shared_ptr_implementation< buffer< T, Dimensions, Allocator >, detail::buffer_waiter< T, Dimensions, Allocator > >::hash().

```
00544                                                                    {
00545     // Forward the hashing to the implementation
00546     return b.hash();
00547   }
```

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- include/CL/sycl/buffer.hpp

10.9 std::hash< cl::sycl::context > Struct Template Reference

```
#include <context.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::context &c) const

10.9.1 Detailed Description

template<>
struct std::hash< cl::sycl::context >

Definition at line 223 of file context.hpp.

10.9.2 Member Function Documentation

10.9.2.1 operator()()

```
auto std::hash< cl::sycl::context >::operator() (
            const cl::sycl::context & c ) const  [inline]
```

Definition at line 224 of file context.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
00224                                                   {
00225     return c.hash();
00226     }
```

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- include/CL/sycl/context.hpp

10.10 std::hash< cl::sycl::device > Struct Template Reference

```
#include <device.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::device &d) const

10.10.1 Detailed Description

template<>
struct std::hash< cl::sycl::device >

Definition at line 312 of file device.hpp.

10.10.2 Member Function Documentation

10.10.2.1 operator()()

```
auto std::hash< cl::sycl::device >::operator() (
            const cl::sycl::device & d ) const  [inline]
```

Definition at line 314 of file device.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
00314                                                {
00315     // Forward the hashing to the implementation
00316     return d.hash();
00317   }
```

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- include/CL/sycl/device.hpp

10.11 std::hash< cl::sycl::kernel > Struct Template Reference

```
#include <kernel.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::kernel &k) const

10.11.1 Detailed Description

template<>
struct std::hash< cl::sycl::kernel >

Definition at line 125 of file kernel.hpp.

10.11.2 Member Function Documentation

10.11.2.1 operator()()

```
auto std::hash< cl::sycl::kernel >::operator() (
            const cl::sycl::kernel & k ) const  [inline]
```

Definition at line 127 of file kernel.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
00127                                                {
00128     // Forward the hashing to the implementation
00129     return k.hash();
00130   }
```

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- include/CL/sycl/kernel.hpp

10.12 std::hash< cl::sycl::platform > Struct Template Reference

```
#include <platform.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::platform &p) const

10.12.1 Detailed Description

template<>
struct std::hash< cl::sycl::platform >

Definition at line 199 of file platform.hpp.

10.12.2 Member Function Documentation

10.12.2.1 operator()()

```
auto std::hash< cl::sycl::platform >::operator() (
            const cl::sycl::platform & p ) const  [inline]
```

Definition at line 201 of file platform.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
00201                                                   {
00202     // Forward the hashing to the implementation
00203     return p.hash();
00204   }
```

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- include/CL/sycl/platform.hpp

10.13 std::hash< cl::sycl::queue > Struct Template Reference

```
#include <queue.hpp>
```

Public Member Functions

- auto operator() (const cl::sycl::queue &q) const

10.13.1 Detailed Description

template<>
struct std::hash< cl::sycl::queue >

Definition at line 387 of file queue.hpp.

10.13.2 Member Function Documentation

10.13.2.1 operator()()

```
auto std::hash< cl::sycl::queue >::operator() (
            const cl::sycl::queue & q ) const  [inline]
```

Definition at line 389 of file queue.hpp.

References cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash().

```
00389                                                {
00390     // Forward the hashing to the implementation
00391     return q.hash();
00392   }
```

Here is the call graph for this function:

The documentation for this struct was generated from the following file:

- include/CL/sycl/queue.hpp

10.14 cl::sycl::detail::host_device Class Reference

SYCL host device.

```
#include <host_device.hpp>
```

Inheritance diagram for cl::sycl::detail::host_device:

Collaboration diagram for cl::sycl::detail::host_device:

Public Member Functions

- cl_device_id get () const override

  Return the cl_device_id of the underlying OpenCL platform.
- boost::compute::device & get_boost_compute () override

  Return the underlying Boost.Compute device.
- bool is_host () const override

  Return true since the device is a SYCL host device.
- bool is_cpu () const override

  Return false since the host device is not an OpenCL CPU device.

- bool is_gpu () const override

  Return false since the host device is not an OpenCL GPU device.
- bool is_accelerator () const override

  Return false since the host device is not an OpenCL accelerator device.
- cl::sycl::platform get_platform () const override

  Return the platform of device.
- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the device.

Additional Inherited Members

10.14.1 Detailed Description

SYCL host device.

Todo The implementation is quite minimal for now. :-)

Definition at line 31 of file host_device.hpp.

10.14.2 Member Function Documentation

10.14.2.1 get()

```
cl_device_id cl::sycl::detail::host_device::get ( ) const  [inline], [override], [virtual]
```

Return the cl_device_id of the underlying OpenCL platform.

This throws an error since there is no OpenCL device associated to the host device.

Implements cl::sycl::detail::device.

Definition at line 42 of file host_device.hpp.

```
00042                                   {
00043     throw non_cl_error("The host device has no OpenCL device");
00044   }
```

10.14.2.2 get_boost_compute()

```
boost::compute::device& cl::sycl::detail::host_device::get_boost_compute ( ) [inline], [override],
[virtual]
```

Return the underlying Boost.Compute device.

This throws an error since there is no OpenCL device associated to the host device.

Implements cl::sycl::detail::device.

Definition at line 52 of file host_device.hpp.

```
00052                                                   {
00053     throw non_cl_error("The host device has no underlying OpenCL device");
00054   }
```

10.14.2.3 get_platform()

```
cl::sycl::platform cl::sycl::detail::host_device::get_platform ( ) const [inline], [override],
[virtual]
```

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

Implements cl::sycl::detail::device.

Definition at line 88 of file host_device.hpp.

References cl::sycl::detail::unimplemented().

```
00088                                                   {
00089     detail::unimplemented();
00090     return {};
00091   }
```

Here is the call graph for this function:

10.14.2.4 has_extension()

```
bool cl::sycl::detail::host_device::has_extension (
            const string_class & extension ) const  [inline], [override], [virtual]
```

Specify whether a specific extension is supported on the device.

Todo To be implemented

Implements cl::sycl::detail::device.

Definition at line 112 of file host_device.hpp.

References cl::sycl::detail::unimplemented().

```
00112                                                                    {
00113     detail::unimplemented();
00114     return {};
00115   }
```

Here is the call graph for this function:

10.14.2.5 is_accelerator()

```
bool cl::sycl::detail::host_device::is_accelerator ( ) const  [inline], [override], [virtual]
```

Return false since the host device is not an OpenCL accelerator device.

Implements cl::sycl::detail::device.

Definition at line 77 of file host_device.hpp.

```
00077                                        {
00078     return false;
00079   }
```

10.14.2.6 is_cpu()

```
bool cl::sycl::detail::host_device::is_cpu ( ) const  [inline], [override], [virtual]
```

Return false since the host device is not an OpenCL CPU device.

Implements cl::sycl::detail::device.

Definition at line 65 of file host_device.hpp.

```
00065                                {
00066     return false;
00067   }
```

10.14.2.7 is_gpu()

```
bool cl::sycl::detail::host_device::is_gpu ( ) const  [inline], [override], [virtual]
```

Return false since the host device is not an OpenCL GPU device.

Implements cl::sycl::detail::device.

Definition at line 71 of file host_device.hpp.

```
00071                                {
00072     return false;
00073   }
```

10.14.2.8 is_host()

```
bool cl::sycl::detail::host_device::is_host ( ) const  [inline], [override], [virtual]
```

Return true since the device is a SYCL host device.

Implements cl::sycl::detail::device.

Definition at line 59 of file host_device.hpp.

```
00059                                 {
00060     return true;
00061   }
```

The documentation for this class was generated from the following file:

- include/CL/sycl/device/detail/host_device.hpp

10.15 cl::sycl::detail::host_queue Class Reference

Some implementation details about the SYCL queue.

```
#include <host_queue.hpp>
```

Inheritance diagram for cl::sycl::detail::host_queue:

Collaboration diagram for cl::sycl::detail::host_queue:

Private Member Functions

- cl_command_queue get () const override

  Return the cl_command_queue of the underlying OpenCL queue.
- boost::compute::command_queue & get_boost_compute () override

Return the underlying Boost.Compute command queue.

- cl::sycl::context get_context () const override

  Return the SYCL host queue's host context.

- cl::sycl::device get_device () const override

  Return the SYCL host device the host queue is associated with.

- bool is_host () const override

  Claim proudly that the queue is executing on the SYCL host device.

Additional Inherited Members

10.15.1 Detailed Description

Some implementation details about the SYCL queue.

Note that a host queue is not a singleton, compared to host device or host platform, for example.

Definition at line 30 of file host_queue.hpp.

10.15.2 Member Function Documentation

10.15.2.1 get()

```
cl_command_queue cl::sycl::detail::host_queue::get ( ) const  [inline], [override], [private],
[virtual]
```

Return the cl_command_queue of the underlying OpenCL queue.

This throws an error since there is no OpenCL queue associated to the host queue.

Implements cl::sycl::detail::queue.

Definition at line 39 of file host_queue.hpp.

```
00039                                       {
00040     throw non_cl_error("The host queue has no OpenCL command queue");
00041   }
```

10.15.2.2 get_boost_compute()

```
boost::compute::command_queue& cl::sycl::detail::host_queue::get_boost_compute ( ) [inline], [override], [private], [virtual]
```

Return the underlying Boost.Compute command queue.

This throws an error since there is no OpenCL queue associated to the host queue.

Implements cl::sycl::detail::queue.

Definition at line 49 of file host_queue.hpp.

```
00049                                                                {
00050     throw non_cl_error("The host queue has no OpenCL command queue");
00051   }
```

10.15.2.3 get_context()

```
cl::sycl::context cl::sycl::detail::host_queue::get_context ( ) const [inline], [override], [private], [virtual]
```

Return the SYCL host queue's host context.

Implements cl::sycl::detail::queue.

Definition at line 56 of file host_queue.hpp.

```
00056                                                  {
00057     // Return the default context which is the host context
00058     return {};
00059   }
```

10.15.2.4 get_device()

```
cl::sycl::device cl::sycl::detail::host_queue::get_device ( ) const [inline], [override], [private], [virtual]
```

Return the SYCL host device the host queue is associated with.

Implements cl::sycl::detail::queue.

Definition at line 63 of file host_queue.hpp.

```
00063                                                  {
00064     // Return the default device which is the host device
00065     return {};
00066   }
```

10.15.2.5 is_host()

```
bool cl::sycl::detail::host_queue::is_host ( ) const  [inline], [override], [private], [virtual]
```

Claim proudly that the queue is executing on the SYCL host device.

Implements cl::sycl::detail::queue.

Definition at line 70 of file host_queue.hpp.

```
00070                                   {
00071     return true;
00072   }
```

The documentation for this class was generated from the following file:

- include/CL/sycl/queue/detail/host_queue.hpp

10.16 cl::sycl::is_wrapper< T > Struct Template Reference

```
#include <opencl_types.hpp>
```

Inheritance diagram for cl::sycl::is_wrapper< T >:

Collaboration diagram for cl::sycl::is_wrapper< T >:

10.16.1 Detailed Description

template<class T>
struct cl::sycl::is_wrapper< T >

Definition at line 81 of file opencl_types.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/opencl_types.hpp

10.17 cl::sycl::xilinx::partition::none Struct Reference

Represent a none partition.

```
#include <partition_array.hpp>
```

Collaboration diagram for cl::sycl::xilinx::partition::none:

Static Public Attributes

- static constexpr auto partition_type = type::none

10.17.1 Detailed Description

Represent a none partition.

The single array would be the same as std::array.

Definition at line 131 of file partition_array.hpp.

10.17.2 Member Data Documentation

10.17.2.1 partition_type

```
constexpr auto cl::sycl::xilinx::partition::none::partition_type = type::none  [static]
```

Definition at line 132 of file partition_array.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/vendor/Xilinx/partition_array.hpp

10.18 cl::sycl::detail::opencl_context Class Reference

SYCL OpenCL context.

```
#include <opencl_context.hpp>
```

Inheritance diagram for cl::sycl::detail::opencl_context:

Collaboration diagram for cl::sycl::detail::opencl_context:

Public Member Functions

- cl_context get () const override

  Return the underlying `cl_context` of the `cl::sycl::context`.
- boost::compute::context & get_boost_compute () override

  Return the underlying `boost::compute::context` of the `cl::sycl::context`.
- boost::compute::command_queue & get_boost_queue () override

  Return the queue that is associated to the context.
- bool is_host () const override

  Return false because the context is not a SYCL host context.
- cl::sycl::platform get_platform () const override

  Return the platform of the context.
- vector_class< cl::sycl::device > get_devices () const override

  Returns the set of devices that are part of this context.
- ~opencl_context () override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_context > instance (const boost::compute::context &c)

  Get a singleton instance of the `opencl_context`.

Private Member Functions

- opencl_context (const boost::compute::context &c)

  Only the instance factory can build it.

Private Attributes

- boost::compute::context c

  User the Boost Compute abstraction of the OpenCL context.
- boost::compute::command_queue q

  A boost `command_queue` associated to an OpenCL context for when we need to transfer data but no queue is given (eg.

Static Private Attributes

- static detail::cache< cl_context, detail::opencl_context > cache

  A cache to always return the same alive context for a given OpenCL context.

10.18.1 Detailed Description

SYCL OpenCL context.

Definition at line 30 of file opencl_context.hpp.

10.18.2 Constructor & Destructor Documentation

10.18.2.1 opencl_context()

```
cl::sycl::detail::opencl_context::opencl_context (
            const boost::compute::context & c )  [inline], [private]
```

Only the instance factory can build it.

Definition at line 122 of file opencl_context.hpp.

References c.

```
00122                                                :
00123     c { c },
00124     q { boost::compute::command_queue { c, c.get_device() } } {}
```

10.18.2.2 ~opencl_context()

```
cl::sycl::detail::opencl_context::~opencl_context ( )  [inline], [override]
```

Unregister from the cache on destruction.

Definition at line 129 of file opencl_context.hpp.

References cache, cl::sycl::detail::cache< Key, Value >::remove(), and TRISYCL_WEAK_ATTRIB_PREFIX.

```
00129                                {
00130     cache.remove(c.get());
00131   }
```

Here is the call graph for this function:

10.18.3 Member Function Documentation

10.18.3.1 get()

```
cl_context cl::sycl::detail::opencl_context::get ( ) const  [inline], [override], [virtual]
```

Return the underlying `cl_context` of the `cl::sycl::context`.

Implements cl::sycl::detail::context.

Definition at line 51 of file opencl_context.hpp.

```
00051                                   {
00052     return c.get();
00053   }
```

10.18.3.2 get_boost_compute()

```
boost::compute::context& cl::sycl::detail::opencl_context::get_boost_compute ( )  [inline],
[override], [virtual]
```

Return the underlying `boost::compute::context` of the `cl::sycl::context`.

Implements cl::sycl::detail::context.

Definition at line 59 of file opencl_context.hpp.

References c.

```
00059                                                   {
00060     return c;
00061   }
```

10.18.3.3 get_boost_queue()

```
boost::compute::command_queue& cl::sycl::detail::opencl_context::get_boost_queue ( )  [inline],
[override], [virtual]
```

Return the queue that is associated to the context.

Implements cl::sycl::detail::context.

Definition at line 65 of file opencl_context.hpp.

References q.

```
00065                                                       {
00066     return q;
00067   }
```

10.18.3.4 get_devices()

```
vector_class<cl::sycl::device> cl::sycl::detail::opencl_context::get_devices ( ) const [inline], [override], [virtual]
```

Returns the set of devices that are part of this context.

Todo To be implemented

Implements cl::sycl::detail::context.

Definition at line 106 of file opencl_context.hpp.

References cl::sycl::detail::unimplemented().

```
00106                                                                   {
00107     detail::unimplemented();
00108     return {};
00109   }
```

Here is the call graph for this function:

10.18.3.5 get_platform()

```
cl::sycl::platform cl::sycl::detail::opencl_context::get_platform ( ) const [inline], [override], [virtual]
```

Return the platform of the context.

Return synchronous errors via the SYCL exception class.

Todo To be implemented

Implements cl::sycl::detail::context.

Definition at line 96 of file opencl_context.hpp.

References cl::sycl::detail::unimplemented().

```
00096                                                   {
00097     detail::unimplemented();
00098     return {};
00099   }
```

Here is the call graph for this function:

10.18.3.6 instance()

```
static std::shared_ptr<opencl_context> cl::sycl::detail::opencl_context::instance (
            const boost::compute::context & c )  [inline], [static]
```

Get a singleton instance of the `opencl_context`.

Definition at line 113 of file opencl_context.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

Referenced by cl::sycl::context::context().

```
00113                                           {
00114     return cache.get_or_register(c.get(),
00115                                  [&] { return new opencl_context {
      c }; });
00116   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

10.18.3.7 is_host()

```
bool cl::sycl::detail::opencl_context::is_host ( ) const  [inline], [override], [virtual]
```

Return false because the context is not a SYCL host context.

Implements cl::sycl::detail::context.

Definition at line 71 of file opencl_context.hpp.

References cl::sycl::detail::unimplemented().

```
00071                                   {
00072     return false;
00073   }
```

Here is the call graph for this function:

10.18.4 Member Data Documentation

10.18.4.1 c

```
boost::compute::context cl::sycl::detail::opencl_context::c  [private]
```

User the Boost Compute abstraction of the OpenCL context.

Definition at line 33 of file opencl_context.hpp.

Referenced by get_boost_compute(), and opencl_context().

10.18.4.2 cache

```
detail::cache<cl_context, detail::opencl_context> cl::sycl::detail::opencl_context::cache
[static], [private]
```

A cache to always return the same alive context for a given OpenCL context.

C++11 guaranties the static construction is thread-safe

Definition at line 46 of file opencl_context.hpp.

Referenced by ~opencl_context().

10.18.4.3 q

```
boost::compute::command_queue cl::sycl::detail::opencl_context::q  [private]
```

A boost `command_queue` associated to an OpenCL context for when we need to transfer data but no queue is given (eg.

When an buffer accessor is created)

Definition at line 39 of file opencl_context.hpp.

Referenced by get_boost_queue().

The documentation for this class was generated from the following file:

- include/CL/sycl/context/detail/opencl_context.hpp

10.19 cl::sycl::detail::opencl_device Class Reference

SYCL OpenCL device.

```
#include <opencl_device.hpp>
```

Inheritance diagram for cl::sycl::detail::opencl_device:

Collaboration diagram for cl::sycl::detail::opencl_device:

Public Member Functions

- cl_device_id get () const override

  Return the cl_device_id of the underlying OpenCL device.
- boost::compute::device & get_boost_compute () override

  Return the underlying Boost.Compute device.
- bool is_host () const override

  Return false since an OpenCL device is not the SYCL host device.
- bool is_cpu () const override

  Test if the OpenCL is a CPU device.
- bool is_gpu () const override

  Test if the OpenCL is a GPU device.
- bool is_accelerator () const override

  Test if the OpenCL is an accelerator device.
- cl::sycl::platform get_platform () const override

Return the platform of device.

- bool has_extension (const string_class &extension) const override

  Specify whether a specific extension is supported on the device.

- ~opencl_device () override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_device > instance (const boost::compute::device &d)

Private Member Functions

- opencl_device (const boost::compute::device &d)

  Only the instance factory can build it.

Private Attributes

- boost::compute::device d

  Use the Boost Compute abstraction of the OpenCL device.

Static Private Attributes

- static detail::cache< cl_device_id, detail::opencl_device > cache

  A cache to always return the same alive device for a given OpenCL device.

10.19.1 Detailed Description

SYCL OpenCL device.

Definition at line 30 of file opencl_device.hpp.

10.19.2 Constructor & Destructor Documentation

10.19.2.1 opencl_device()

```
cl::sycl::detail::opencl_device::opencl_device (
            const boost::compute::device & d )  [inline], [private]
```

Only the instance factory can build it.

Definition at line 126 of file opencl_device.hpp.

```
00126 : d { d } {}
```

10.19.2.2 ~opencl_device()

```
cl::sycl::detail::opencl_device::~opencl_device ( )  [inline], [override]
```

Unregister from the cache on destruction.

Definition at line 131 of file opencl_device.hpp.

References cache, cl::sycl::detail::cache< Key, Value >::remove(), TRISYCL_WEAK_ATTRIB_PREFIX, and cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00131                                {
00132     cache.remove(d.id());
00133   }
```

Here is the call graph for this function:

10.19.3 Member Function Documentation

10.19.3.1 get()

```
cl_device_id cl::sycl::detail::opencl_device::get ( ) const  [inline], [override], [virtual]
```

Return the cl_device_id of the underlying OpenCL device.

Implements cl::sycl::detail::device.

Definition at line 45 of file opencl_device.hpp.

```
00045                                        {
00046     return d.id();
00047   }
```

10.19.3.2 get_boost_compute()

```
boost::compute::device& cl::sycl::detail::opencl_device::get_boost_compute ( ) [inline], [override], [virtual]
```

Return the underlying Boost.Compute device.

Implements cl::sycl::detail::device.

Definition at line 51 of file opencl_device.hpp.

References d.

```
00051                                                   {
00052     return d;
00053   }
```

10.19.3.3 get_platform()

```
cl::sycl::platform cl::sycl::detail::opencl_device::get_platform ( ) const [inline], [override], [virtual]
```

Return the platform of device.

Return synchronous errors via the SYCL exception class.

Implements cl::sycl::detail::device.

Definition at line 87 of file opencl_device.hpp.

References cl::sycl::platform::platform(), and cl::sycl::detail::unimplemented().

```
00087                                                   {
00088     return d.platform();
00089   }
```

Here is the call graph for this function:

10.19.3.4 has_extension()

```
bool cl::sycl::detail::opencl_device::has_extension (
            const string_class & extension ) const  [inline], [override], [virtual]
```

Specify whether a specific extension is supported on the device.

Todo To be implemented

Implements cl::sycl::detail::device.

Definition at line 110 of file opencl_device.hpp.

References cl::sycl::detail::unimplemented().

```
00110                                                                   {
00111     detail::unimplemented();
00112     return {};
00113   }
```

Here is the call graph for this function:

10.19.3.5 instance()

```
static std::shared_ptr<opencl_device> cl::sycl::detail::opencl_device::instance (
            const boost::compute::device & d )  [inline], [static]
```

Definition at line 118 of file opencl_device.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

Referenced by cl::sycl::device::device().

```
00118                                       {
00119     return cache.get_or_register(d.id(),
00120                                  [&] { return new opencl_device { d }; });
00121   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

10.19.3.6 is_accelerator()

```
bool cl::sycl::detail::opencl_device::is_accelerator ( ) const  [inline], [override], [virtual]
```

Test if the OpenCL is an accelerator device.

Implements cl::sycl::detail::device.

Definition at line 77 of file opencl_device.hpp.

```
00077                                         {
00078     // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00079     return d.type() & boost::compute::device::accelerator;
00080   }
```

10.19.3.7 is_cpu()

```
bool cl::sycl::detail::opencl_device::is_cpu ( ) const  [inline], [override], [virtual]
```

Test if the OpenCL is a CPU device.

Implements cl::sycl::detail::device.

Definition at line 63 of file opencl_device.hpp.

```
00063                                 {
00064     // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00065     return d.type() & boost::compute::device::cpu;
00066   }
```

10.19.3.8 is_gpu()

```
bool cl::sycl::detail::opencl_device::is_gpu ( ) const  [inline], [override], [virtual]
```

Test if the OpenCL is a GPU device.

Implements cl::sycl::detail::device.

Definition at line 70 of file opencl_device.hpp.

```
00070                                {
00071     // Even in Boost.Compute the type is a bit-field, so use & instead of ==
00072     return d.type() & boost::compute::device::gpu;
00073   }
```

10.19.3.9 is_host()

```
bool cl::sycl::detail::opencl_device::is_host ( ) const  [inline], [override], [virtual]
```

Return false since an OpenCL device is not the SYCL host device.

Implements cl::sycl::detail::device.

Definition at line 57 of file opencl_device.hpp.

```
00057                                {
00058     return false;
00059   }
```

10.19.4 Member Data Documentation

10.19.4.1 cache

```
detail::cache<cl_device_id, detail::opencl_device> cl::sycl::detail::opencl_device::cache
[static], [private]
```

A cache to always return the same alive device for a given OpenCL device.

C++11 guaranties the static construction is thread-safe

Definition at line 40 of file opencl_device.hpp.

Referenced by ~opencl_device().

10.19.4.2 d

```
boost::compute::device cl::sycl::detail::opencl_device::d [private]
```

Use the Boost Compute abstraction of the OpenCL device.

Definition at line 33 of file opencl_device.hpp.

Referenced by get_boost_compute().

The documentation for this class was generated from the following file:

- include/CL/sycl/device/detail/opencl_device.hpp

10.20 cl::sycl::detail::opencl_kernel Class Reference

An abstraction of the OpenCL kernel.

```
#include <opencl_kernel.hpp>
```

Inheritance diagram for cl::sycl::detail::opencl_kernel:

Collaboration diagram for cl::sycl::detail::opencl_kernel:

Public Member Functions

- cl_kernel get () const override

  Return the underlying OpenCL object.
- boost::compute::kernel & get_boost_compute () override

  Return the Boost.Compute OpenCL kernel object for this kernel.
- void single_task (std::shared_ptr< detail::task > task, std::shared_ptr< detail::queue > q) override

  Launch a single task of the OpenCL kernel.
- TRISYCL_ParallelForKernel_RANGE (1) TRISYCL_ParallelForKernel_RANGE(2) TRISYCL_ParallelFor↩ Kernel_RANGE(3) ~opencl_kernel() override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_kernel > instance (const boost::compute::kernel &k)

Private Member Functions

- opencl_kernel (const boost::compute::kernel &k)

Private Attributes

- boost::compute::kernel k

  Use the Boost Compute abstraction of the OpenCL kernel.

Static Private Attributes

- static detail::cache< cl_kernel, detail::opencl_kernel > cache

  A cache to always return the same alive kernel for a given OpenCL kernel.

10.20.1 Detailed Description

An abstraction of the OpenCL kernel.

Definition at line 29 of file opencl_kernel.hpp.

10.20.2 Constructor & Destructor Documentation

10.20.2.1 opencl_kernel()

```
cl::sycl::detail::opencl_kernel::opencl_kernel (
            const boost::compute::kernel & k )  [inline], [private]
```

Definition at line 42 of file opencl_kernel.hpp.

```
00042 : k { k } {}
```

10.20.3 Member Function Documentation

10.20.3.1 get()

```
cl_kernel cl::sycl::detail::opencl_kernel::get ( ) const  [inline], [override], [virtual]
```

Return the underlying OpenCL object.

Todo Improve the spec to deprecate C OpenCL host API and move to C++ instead to avoid this ugly ownership management

Todo Test error and throw. Externalize this feature in Boost.Compute?

Implements cl::sycl::detail::kernel.

Definition at line 58 of file opencl_kernel.hpp.

```
00058                                {
00059     /// \todo Test error and throw. Externalize this feature in Boost.Compute?
00060     clRetainKernel(k);
00061     return k.get();
00062   }
```

10.20.3.2 get_boost_compute()

```
boost::compute::kernel& cl::sycl::detail::opencl_kernel::get_boost_compute ( ) [inline],
[override], [virtual]
```

Return the Boost.Compute OpenCL kernel object for this kernel.

This is an extension.

Implements cl::sycl::detail::kernel.

Definition at line 69 of file opencl_kernel.hpp.

References k, and cl::sycl::detail::unimplemented().

```
00069                                                    {
00070     return k;
00071   }
```

Here is the call graph for this function:

10.20.3.3 instance()

```
static std::shared_ptr<opencl_kernel> cl::sycl::detail::opencl_kernel::instance (
            const boost::compute::kernel & k ) [inline], [static]
```

Definition at line 48 of file opencl_kernel.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

Referenced by cl::sycl::kernel::kernel().

```
00048                                        {
00049     return cache.get_or_register(k.get(),
00050                                  [&] { return new opencl_kernel { k }; });
00051   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

10.20.3.4 single_task()

```
void cl::sycl::detail::opencl_kernel::single_task (
            std::shared_ptr< detail::task > task,
            std::shared_ptr< detail::queue > q )  [inline], [override], [virtual]
```

Launch a single task of the OpenCL kernel.

Todo Remove either task or q

Implements cl::sycl::detail::kernel.

Definition at line 91 of file opencl_kernel.hpp.

```
00092                                                        {
00093     q->get_boost_compute().enqueue_task(k);
00094     /* For now use a crude synchronization mechanism to map directly a
00095        host task to an accelerator task */
00096     q->get_boost_compute().finish();
00097   }
```

10.20.3.5 TRISYCL_ParallelForKernel_RANGE()

```
cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_RANGE (
            1  ) [inline], [override]
```

Unregister from the cache on destruction.

Definition at line 126 of file opencl_kernel.hpp.

References cache, cl::sycl::detail::cache< Key, Value >::remove(), TRISYCL_WEAK_ATTRIB_PREFIX, and cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00133                                {
00134     cache.remove(k.get());
00135   }
```

Here is the call graph for this function:

10.20.4 Member Data Documentation

10.20.4.1 cache

```
detail::cache<cl_kernel, detail::opencl_kernel> cl::sycl::detail::opencl_kernel::cache [static],
[private]
```

A cache to always return the same alive kernel for a given OpenCL kernel.

C++11 guaranties the static construction is thread-safe

Definition at line 40 of file opencl_kernel.hpp.

Referenced by TRISYCL_ParallelForKernel_RANGE().

10.20.4.2 k

```
boost::compute::kernel cl::sycl::detail::opencl_kernel::k  [private]
```

Use the Boost Compute abstraction of the OpenCL kernel.

Definition at line 33 of file opencl_kernel.hpp.

Referenced by get_boost_compute().

The documentation for this class was generated from the following file:

- include/CL/sycl/kernel/detail/opencl_kernel.hpp

10.21 cl::sycl::detail::opencl_queue Class Reference

Some implementation details about the SYCL queue.

```
#include <opencl_queue.hpp>
```

Inheritance diagram for cl::sycl::detail::opencl_queue:

Collaboration diagram for cl::sycl::detail::opencl_queue:

Public Member Functions

- ~opencl_queue () override

  Unregister from the cache on destruction.

Static Public Member Functions

- static std::shared_ptr< opencl_queue > instance (const boost::compute::command_queue &q)

  Get a singleton instance of the opencl_queue.
- static std::shared_ptr< detail::queue > instance (const cl::sycl::device &d)

  Create a new queue associated to this device.

Private Member Functions

- cl_command_queue get () const override

  Return the cl_command_queue of the underlying OpenCL queue.
- boost::compute::command_queue & get_boost_compute () override

  Return the underlying Boost.Compute command queue.
- cl::sycl::context get_context () const override

  Return the SYCL context associated to the queue.
- cl::sycl::device get_device () const override

  Return the SYCL device associated to the queue.
- bool is_host () const override

  Claim proudly that an OpenCL queue cannot be the SYCL host queue.
- opencl_queue (const boost::compute::command_queue &q)

  Only the instance factory can built it.

Private Attributes

- boost::compute::command_queue q

  Use the Boost Compute abstraction of the OpenCL command queue.

Static Private Attributes

- static detail::cache< cl_command_queue, detail::opencl_queue > cache

  A cache to always return the same alive queue for a given OpenCL command queue.

Additional Inherited Members

10.21.1 Detailed Description

Some implementation details about the SYCL queue.

Definition at line 23 of file opencl_queue.hpp.

10.21.2 Constructor & Destructor Documentation

10.21.2.1 opencl_queue()

```
cl::sycl::detail::opencl_queue::opencl_queue (
            const boost::compute::command_queue & q )  [inline], [private]
```

Only the instance factory can built it.

Definition at line 67 of file opencl_queue.hpp.

```
00067 : q { q } {}
```

10.21.2.2 ~opencl_queue()

```
cl::sycl::detail::opencl_queue::~opencl_queue ( )  [inline], [override]
```

Unregister from the cache on destruction.

Definition at line 96 of file opencl_queue.hpp.

References cache, cl::sycl::detail::cache< Key, Value >::remove(), TRISYCL_WEAK_ATTRIB_PREFIX, and cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00096                           {
00097     cache.remove(q.get());
00098   }
```

Here is the call graph for this function:

10.21.3 Member Function Documentation

10.21.3.1 get()

```
cl_command_queue cl::sycl::detail::opencl_queue::get ( ) const [inline], [override], [private],
[virtual]
```

Return the cl_command_queue of the underlying OpenCL queue.

Implements cl::sycl::detail::queue.

Definition at line 36 of file opencl_queue.hpp.

```
00036                                        {
00037     return q.get();
00038   }
```

10.21.3.2 get_boost_compute()

```
boost::compute::command_queue& cl::sycl::detail::opencl_queue::get_boost_compute ( ) [inline],
[override], [private], [virtual]
```

Return the underlying Boost.Compute command queue.

Implements cl::sycl::detail::queue.

Definition at line 42 of file opencl_queue.hpp.

References q.

```
00042                                                        {
00043     return q;
00044   }
```

10.21.3.3 get_context()

```
cl::sycl::context cl::sycl::detail::opencl_queue::get_context ( ) const [inline], [override],
[private], [virtual]
```

Return the SYCL context associated to the queue.

Implements cl::sycl::detail::queue.

Definition at line 48 of file opencl_queue.hpp.

```
00048                                            {
00049     return q.get_context();
00050   }
```

10.21.3.4 get_device()

```
cl::sycl::device cl::sycl::detail::opencl_queue::get_device ( ) const  [inline], [override],
[private], [virtual]
```

Return the SYCL device associated to the queue.

Implements cl::sycl::detail::queue.

Definition at line 54 of file opencl_queue.hpp.

```
00054                                         {
00055     return q.get_device();
00056   }
```

10.21.3.5 instance() [1/2]

```
static std::shared_ptr<opencl_queue> cl::sycl::detail::opencl_queue::instance (
            const boost::compute::command_queue & q )  [inline], [static]
```

Get a singleton instance of the opencl_queue.

Definition at line 73 of file opencl_queue.hpp.

References cl::sycl::detail::cache< Key, Value >::get_or_register().

Referenced by instance(), and cl::sycl::queue::queue().

```
00073                                                 {
00074     return cache.get_or_register(q.get(),
00075                                  [&] { return new opencl_queue { q }; });
00076   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

10.21.3.6 instance() [2/2]

```
static std::shared_ptr<detail::queue> cl::sycl::detail::opencl_queue::instance (
            const cl::sycl::device & d )  [inline], [static]
```

Create a new queue associated to this device.

Todo Check with SYCL committee what is the expected behaviour here about the context. Is this a new context everytime, or always the same for a given device?

Definition at line 86 of file opencl_queue.hpp.

References cl::sycl::device::get_boost_compute(), and instance().

```
00086                                   {
00087     return instance (boost::compute::command_queue {
00088         // For now, create a new context every time
00089         boost::compute::context { d.get_boost_compute() },
00090         d.get_boost_compute()
00091           });
00092   }
```

Here is the call graph for this function:

10.21.3.7 is_host()

```
bool cl::sycl::detail::opencl_queue::is_host ( ) const  [inline], [override], [private], [virtual]
```

Claim proudly that an OpenCL queue cannot be the SYCL host queue.

Implements cl::sycl::detail::queue.

Definition at line 60 of file opencl_queue.hpp.

```
00060                                   {
00061     return false;
00062   }
```

10.21.4 Member Data Documentation

10.21.4.1 cache

```
detail::cache<cl_command_queue, detail::opencl_queue> cl::sycl::detail::opencl_queue::cache
[static], [private]
```

A cache to always return the same alive queue for a given OpenCL command queue.

C++11 guaranties the static construction is thread-safe

Definition at line 33 of file opencl_queue.hpp.

Referenced by ~opencl_queue().

10.21.4.2 q

```
boost::compute::command_queue cl::sycl::detail::opencl_queue::q  [private]
```

Use the Boost Compute abstraction of the OpenCL command queue.

Definition at line 26 of file opencl_queue.hpp.

Referenced by get_boost_compute().

The documentation for this class was generated from the following file:

- include/CL/sycl/queue/detail/opencl_queue.hpp

10.22 cl::sycl::info::param_traits< T, Param > Struct Template Reference

Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

```
#include <param_traits.hpp>
```

10.22.1 Detailed Description

template<typename T, T Param>
struct cl::sycl::info::param_traits< T, Param >

Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Definition at line 20 of file param_traits.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/info/param_traits.hpp

10.23 cl::sycl::drt::code::program Struct Reference

A binary program containing some kernels.

`#include <device_runtime.hpp>`

Collaboration diagram for cl::sycl::drt::code::program:

Public Member Functions

- program (std::size_t binary_size, const char *binary)

Public Attributes

- std::size_t binary_size

  The size of a binary program.
- unsigned const char * binary

  The bytes of program. Use this type for `boost::compute`.

Static Public Attributes

- static boost::optional< program > p

  The current program.

10.23.1 Detailed Description

A binary program containing some kernels.

Definition at line 160 of file device_runtime.hpp.

10.23.2 Constructor & Destructor Documentation

10.23.2.1 program()

```
cl::sycl::drt::code::program::program (
            std::size_t binary_size,
            const char * binary )  [inline]
```

Definition at line 172 of file device_runtime.hpp.

References p, TRISYCL_DUMP_T, TRISYCL_WEAK_ATTRIB_PREFIX, and TRISYCL_WEAK_ATTRIB_SUFFIX.

```
00173       : binary_size { binary_size }
00174       , binary { reinterpret_cast<unsigned const char *>(binary) } {
00175         p = *this;
00176         TRISYCL_DUMP_T("Create program with binary size = 0x" <<
      binary_size);
00177       }
```

10.23.3 Member Data Documentation

10.23.3.1 binary

```
unsigned const char* cl::sycl::drt::code::program::binary
```

The bytes of program. Use this type for `boost::compute`.

Definition at line 164 of file device_runtime.hpp.

10.23.3.2 binary_size

```
std::size_t cl::sycl::drt::code::program::binary_size
```

The size of a binary program.

Definition at line 162 of file device_runtime.hpp.

10.23.3.3 p

```
TRISYCL_WEAK_ATTRIB_PREFIX boost::optional< program > TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl↩
::drt::code::program::p  [static]
```

The current program.

Todo Deal with several programs

Definition at line 170 of file device_runtime.hpp.

Referenced by cl::sycl::drt::set_kernel().

The documentation for this struct was generated from the following file:

- include/CL/sycl/device_runtime.hpp

10.24 cl::sycl::detail::queue Struct Reference

Some implementation details about the SYCL queue.

```
#include <queue.hpp>
```

Inheritance diagram for cl::sycl::detail::queue:

Collaboration diagram for cl::sycl::detail::queue:

Public Member Functions

- queue ()

  Initialize the queue with 0 running kernel.
- void wait_for_kernel_execution ()

  Wait for all kernel completion.
- void kernel_start ()

  Signal that a new kernel started on this queue.
- void kernel_end ()

  Signal that a new kernel finished on this queue.

- virtual cl_command_queue get () const =0

  Return the underlying OpenCL command queue after doing a retain.
- virtual boost::compute::command_queue & get_boost_compute ()=0

  Return the underlying Boost.Compute command queue.
- virtual cl::sycl::context get_context () const =0

  Return the SYCL queue's context.
- virtual cl::sycl::device get_device () const =0

  Return the SYCL device the queue is associated with.
- virtual bool is_host () const =0

  Return whether the queue is executing on a SYCL host device.
- virtual ~queue ()

  Wait for all kernel completion before the queue destruction.

Public Attributes

- std::atomic< size_t > running_kernels

  Track the number of kernels still running to wait for their completion.
- std::condition_variable finished

  To signal when all the kernels have completed.
- std::mutex finished_mutex

  To protect the access to the condition variable.

10.24.1 Detailed Description

Some implementation details about the SYCL queue.

Definition at line 30 of file queue.hpp.

10.24.2 Constructor & Destructor Documentation

10.24.2.1 queue()

```
cl::sycl::detail::queue::queue ( ) [inline]
```

Initialize the queue with 0 running kernel.

Definition at line 41 of file queue.hpp.

```
00041         {
00042     running_kernels = 0;
00043   }
```

10.24.2.2 ~queue()

```
virtual cl::sycl::detail::queue::~queue ( )  [inline], [virtual]
```

Wait for all kernel completion before the queue destruction.

Todo Update according spec since queue destruction is non blocking

Definition at line 119 of file queue.hpp.

References wait_for_kernel_execution().

```
00119                    {
00120      wait_for_kernel_execution();
00121    }
```

Here is the call graph for this function:

10.24.3 Member Function Documentation

10.24.3.1 get()

```
virtual cl_command_queue cl::sycl::detail::queue::get ( ) const  [pure virtual]
```

Return the underlying OpenCL command queue after doing a retain.

This memory object is expected to be released by the developer.

Retain a reference to the returned cl_command_queue object.

Caller should release it when finished.

If the queue is a SYCL host queue then an exception is thrown.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

10.24.3.2 get_boost_compute()

```
virtual boost::compute::command_queue& cl::sycl::detail::queue::get_boost_compute ( ) [pure virtual]
```

Return the underlying Boost.Compute command queue.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:

10.24.3.3 get_context()

```
virtual cl::sycl::context cl::sycl::detail::queue::get_context ( ) const [pure virtual]
```

Return the SYCL queue's context.

Report errors using SYCL exception classes.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:

10.24.3.4 get_device()

```
virtual cl::sycl::device cl::sycl::detail::queue::get_device ( ) const  [pure virtual]
```

Return the SYCL device the queue is associated with.

Report errors using SYCL exception classes.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:

10.24.3.5 is_host()

```
virtual bool cl::sycl::detail::queue::is_host ( ) const  [pure virtual]
```

Return whether the queue is executing on a SYCL host device.

Implemented in cl::sycl::detail::host_queue, and cl::sycl::detail::opencl_queue.

Referenced by kernel_end().

Here is the caller graph for this function:

10.24.3.6 kernel_end()

```
void cl::sycl::detail::queue::kernel_end ( )  [inline]
```

Signal that a new kernel finished on this queue.

Definition at line 66 of file queue.hpp.

References get_boost_compute(), get_context(), get_device(), is_host(), and TRISYCL_DUMP_T.

```
00066                    {
00067     TRISYCL_DUMP_T("A kernel of the queue ended");
00068     if (--running_kernels == 0) {
00069       /* It was the last kernel running, so signal the queue just in
00070          case it was working for it for completion
00071
00072          In some cases several threads might want to wait for the
00073          same queue, because of this \c notify_one is not be enough
00074          and a \c notify_all is needed
00075       */
00076       finished.notify_all();
00077     }
00078   }
```

Here is the call graph for this function:

10.24.3.7 kernel_start()

```
void cl::sycl::detail::queue::kernel_start ( )  [inline]
```

Signal that a new kernel started on this queue.

Definition at line 58 of file queue.hpp.

References running_kernels, and TRISYCL_DUMP_T.

```
00058                    {
00059     TRISYCL_DUMP_T("A kernel has been added to the queue");
00060     // One more kernel
00061     ++running_kernels;
00062   }
```

10.24.3.8 wait_for_kernel_execution()

```
void cl::sycl::detail::queue::wait_for_kernel_execution ( )  [inline]
```

Wait for all kernel completion.

Definition at line 47 of file queue.hpp.

References TRISYCL_DUMP_T.

Referenced by ~queue().

```
00047                                   {
00048     TRISYCL_DUMP_T("Queue waiting for kernel completion");
00049     std::unique_lock<std::mutex> ul { finished_mutex };
00050     finished.wait(ul, [&] {
00051         // When there is no kernel running in this queue, we are ready to go
00052         return running_kernels == 0;
00053       });
00054   }
```

Here is the caller graph for this function:

10.24.4 Member Data Documentation

10.24.4.1 finished

```
std::condition_variable cl::sycl::detail::queue::finished
```

To signal when all the kernels have completed.

Definition at line 35 of file queue.hpp.

10.24.4.2 finished_mutex

```
std::mutex cl::sycl::detail::queue::finished_mutex
```

To protect the access to the condition variable.

Definition at line 37 of file queue.hpp.

10.24.4.3 running_kernels

```
std::atomic<size_t> cl::sycl::detail::queue::running_kernels
```

Track the number of kernels still running to wait for their completion.

Definition at line 32 of file queue.hpp.

Referenced by kernel_start().

The documentation for this struct was generated from the following file:

- include/CL/sycl/queue/detail/queue.hpp

10.25 cl::sycl::detail::shared_ptr_implementation< Parent, Implementation > Struct Template Reference

Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

```
#include <shared_ptr_implementation.hpp>
```

Inheritance diagram for cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >:

Collaboration diagram for cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >:

Public Member Functions

- shared_ptr_implementation (std::shared_ptr< Implementation > i)

  The implementation directly as a shared pointer.
- shared_ptr_implementation (Implementation *i)

  The implementation takes the ownership from a raw pointer.
- shared_ptr_implementation ()=default

  Keep all other constructors to have usual shared_ptr behaviour.
- bool operator== (const Parent &other) const

  Equality operator.
- bool operator< (const Parent &other) const

  Inferior operator.
- auto hash () const

  Forward the hashing for unordered containers to the implementation.

Public Attributes

- std::shared_ptr< Implementation > implementation

  The implementation forward everything to this... implementation.

10.25.1 Detailed Description

template<typename Parent, typename Implementation>
struct cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >

Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

To be used, a Parent class wanting an Implementation needs to inherit from.

The implementation ends up in a member really named "implementation".

```
public detail::shared_ptr_implementation<Parent, Implementation>
```

and also inject in std namespace a specialization for

```
hash<Parent>
```

Definition at line 40 of file shared_ptr_implementation.hpp.

10.25.2 Constructor & Destructor Documentation

10.25.2.1 shared_ptr_implementation() [1/3]

```
template<typename Parent, typename Implementation>
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::shared_ptr_implementation
(
            std::shared_ptr< Implementation > i )  [inline]
```

The implementation directly as a shared pointer.

Definition at line 46 of file shared_ptr_implementation.hpp.

```
00047     : implementation { i } {}
```

10.25.2.2 shared_ptr_implementation() [2/3]

```
template<typename Parent, typename Implementation>
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::shared_ptr_implementation
(
            Implementation * i )  [inline]
```

The implementation takes the ownership from a raw pointer.

Definition at line 51 of file shared_ptr_implementation.hpp.

```
00051 : implementation { i } {}
```

10.25.2.3 shared_ptr_implementation() [3/3]

```
template<typename Parent, typename Implementation>
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::shared_ptr_implementation
( )  [default]
```

Keep all other constructors to have usual shared_ptr behaviour.

Referenced by cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >↩ ::shared_ptr_implementation().

Here is the caller graph for this function:

10.25.3 Member Function Documentation

10.25.3.1 hash()

```
template<typename Parent, typename Implementation>
auto cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::hash ( ) const
[inline]
```

Forward the hashing for unordered containers to the implementation.

Definition at line 83 of file shared_ptr_implementation.hpp.

Referenced by std::hash< cl::sycl::kernel >::operator()(), std::hash< cl::sycl::platform >::operator()(), std::hash< cl::sycl::context >::operator()(), std::hash< cl::sycl::device >::operator()(), and std::hash< cl::sycl::queue >↩ ::operator()().

```
00083                    {
00084     return std::hash<decltype(implementation)>{}(implementation);
00085   }
```

Here is the caller graph for this function:

10.25.3.2 operator<()

```
template<typename Parent, typename Implementation>
bool cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::operator< (
            const Parent & other ) const  [inline]
```

Inferior operator.

This is generalized by boost::less_than_comparable from boost::totally_ordered to implement the equality comparable concept

Todo Add this to the spec

Definition at line 77 of file shared_ptr_implementation.hpp.

```
00077                                                {
00078     return implementation < other.implementation;
00079   }
```

10.25.3.3 operator==()

```
template<typename Parent, typename Implementation>
bool cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >::operator== (
            const Parent & other ) const  [inline]
```

Equality operator.

This is generalized by boost::equality_comparable from boost::totally_ordered to implement the equality comparable concept

Definition at line 64 of file shared_ptr_implementation.hpp.

```
00064                                                 {
00065     return implementation == other.implementation;
00066   }
```

10.25.4 Member Data Documentation

10.25.4.1 implementation

```
template<typename Parent, typename Implementation>
std::shared_ptr<Implementation> cl::sycl::detail::shared_ptr_implementation< Parent, Implementation
>::implementation
```

The implementation forward everything to this... implementation.

Definition at line 43 of file shared_ptr_implementation.hpp.

Referenced by cl::sycl::detail::shared_ptr_implementation< static_pipe< T, Capacity >, detail::pipe< T > >↩ ::hash(), and cl::sycl::handler::single_task().

The documentation for this struct was generated from the following file:

- include/CL/sycl/detail/shared_ptr_implementation.hpp

10.26 cl::sycl::detail::singleton< T > Struct Template Reference

Provide a singleton factory.

```
#include <singleton.hpp>
```

Static Public Member Functions

- static std::shared_ptr< T > instance ()

  Get a singleton instance of T.

10.26.1 Detailed Description

template<typename T>
struct cl::sycl::detail::singleton< T >

Provide a singleton factory.

Definition at line 25 of file singleton.hpp.

10.26.2 Member Function Documentation

10.26.2.1 instance()

```
template<typename T>
static std::shared_ptr<T> cl::sycl::detail::singleton< T >::instance ( )  [inline], [static]
```

Get a singleton instance of T.

Use a null_deleter since the singleton should not be deleted, as allocated in the static area

Definition at line 28 of file singleton.hpp.

```
00028                                   {
00029     // C++11 guaranties the static construction is thread-safe
00030     static T single;
00031     /** Use a null_deleter since the singleton should not be deleted,
00032         as allocated in the static area */
00033     static std::shared_ptr<T> sps { &single,
00034                                     boost::null_deleter {} };
00035
00036     return sps;
00037   }
```

The documentation for this struct was generated from the following file:

- include/CL/sycl/detail/singleton.hpp

10.27 cl::sycl::detail::task Struct Reference

The abstraction to represent SYCL tasks executing inside command_group.

```
#include <task.hpp>
```

Inheritance diagram for cl::sycl::detail::task:

Collaboration diagram for cl::sycl::detail::task:

Public Member Functions

- task (const std::shared_ptr< detail::queue > &q)

  Create a task from a submitting queue.
- void schedule (std::function< void(void)> f)

  Add a new task to the task graph and schedule for execution.
- void wait_for_producers ()

  Wait for the required producer tasks to be ready.
- void release_buffers ()

  Release the buffers that have been used by this task.

- void notify_consumers ()

  Notify the waiting tasks that we are done.
- void wait ()

  Wait for this task to be ready.
- void add_buffer (std::shared_ptr< detail::buffer_base > &buf, bool is_write_mode)

  Register a buffer to this task.
- void prelude ()

  Execute the prologues.
- void postlude ()

  Execute the epilogues.
- void add_prelude (const std::function< void(void)> &f)

  Add a function to the prelude to run before kernel execution.
- void add_postlude (const std::function< void(void)> &f)

  Add a function to the postlude to run after kernel execution.
- auto get_queue ()

  Get the queue behind the task to run a kernel on.
- void set_kernel (const std::shared_ptr< cl::sycl::detail::kernel > &k)

  Set the OpenCL-compatible kernel running this task if any.
- cl::sycl::detail::kernel & get_kernel ()

  Get the kernel running if any.
- void set_arg (int arg_index, std::size_t arg_size, const void *scalar_value)

  Set a kernel argument by address.
- template<typename T >
  void set_arg (int arg_index, const T &scalar_value)

  Set a kernel argument by value.
- auto get_compute_buffer (std::size_t order)

  Get the Boost.Compute buffer for an accessor of the task.
- std::size_t register_accessor (std::weak_ptr< detail::accessor_base > a)

  Register an accessor and return its registration order.

Public Attributes

- std::vector< std::shared_ptr< detail::buffer_base > > buffers_in_use

  List of the buffers used by this task.
- std::vector< std::shared_ptr< detail::task > > producer_tasks

  The tasks producing the buffers used by this task.
- std::vector< std::function< void(void)> > prologues

  Keep track of any prologue to be executed before the kernel.
- std::vector< std::function< void(void)> > epilogues

  Keep track of any epilogue to be executed after the kernel.
- bool execution_ended = false

  Store if the execution ended, to be notified by task_ready.
- std::condition_variable ready

  To signal when this task is ready.
- std::mutex ready_mutex

  To protect the access to the condition variable.
- std::shared_ptr< detail::queue > owner_queue

  Keep track of the queue used to submission to notify kernel completion or to run OpenCL kernels on.
- std::shared_ptr< detail::kernel > kernel

  The OpenCL-compatible kernel run by this task, if any.
- std::vector< std::weak_ptr< detail::accessor_base > > accessors

  The accessors indexed by their creation order.

10.27.1 Detailed Description

The abstraction to represent SYCL tasks executing inside command_group.

"enable_shared_from_this" allows to access the shared_ptr behind the scene.

Definition at line 37 of file task.hpp.

10.27.2 Constructor & Destructor Documentation

10.27.2.1 task()

```
cl::sycl::detail::task::task (
            const std::shared_ptr< detail::queue > & q )  [inline]
```

Create a task from a submitting queue.

Definition at line 80 of file task.hpp.

```
00081     : owner_queue { q } {}
```

10.27.3 Member Function Documentation

10.27.3.1 add_buffer()

```
void cl::sycl::detail::task::add_buffer (
            std::shared_ptr< detail::buffer_base > & buf,
            bool is_write_mode )  [inline]
```

Register a buffer to this task.

This is how the dependency graph is incrementally built.

Definition at line 180 of file task.hpp.

References TRISYCL_DUMP_T.

```
00181                                       {
00182     TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
00183     /* Keep track of the use of the buffer to notify its release at
00184        the end of the execution */
00185     buffers_in_use.push_back(buf);
00186     // To be sure the buffer does not disappear before the kernel can run
00187     buf->use();
00188
00189     std::shared_ptr<detail::task> latest_producer;
00190     if (is_write_mode) {
00191       /* Set this task as the latest producer of the buffer so that
00192          another kernel may wait on this task */
00193       latest_producer = buf->set_latest_producer(shared_from_this());
00194     }
00195     else
00196       latest_producer = buf->get_latest_producer();
00197
00198     /* If the buffer is to be produced by a task, add the task in the
00199        producer list to wait on it before running the task core
00200
00201        If a buffer is accessed first in write mode and then in read mode,
00202        the task will add itself as a producer and will wait for itself
00203        when calling \c wait_for_producers, we avoid this by checking that
00204        \c latest_producer is not \c this
00205     */
00206     if (latest_producer && latest_producer != shared_from_this())
00207       producer_tasks.push_back(latest_producer);
00208   }
```

10.27.3.2 add_postlude()

```
void cl::sycl::detail::task::add_postlude (
            const std::function< void(void)> & f )  [inline]
```

Add a function to the postlude to run after kernel execution.

Definition at line 242 of file task.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor().

```
00242                                                   {
00243     epilogues.push_back(f);
00244   }
```

Here is the caller graph for this function:

10.27.3.3 add_prelude()

```
void cl::sycl::detail::task::add_prelude (
            const std::function< void(void)> & f )  [inline]
```

Add a function to the prelude to run before kernel execution.

Definition at line 234 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor().

```
00234                                                   {
00235     TRISYCL_DUMP_T("task::add_prelude");
00236
00237     prologues.push_back(f);
00238   }
```

Here is the caller graph for this function:

10.27.3.4 get_compute_buffer()

```
auto cl::sycl::detail::task::get_compute_buffer (
            std::size_t order )  [inline]
```

Get the Boost.Compute buffer for an accessor of the task.

Definition at line 297 of file task.hpp.

Referenced by cl::sycl::drt::serialize_accessor_arg().

```
00297                                          {
00298     return accessors[order].lock()->get_cl_buffer();
00299   }
```

Here is the caller graph for this function:

10.27.3.5 get_kernel()

```
cl::sycl::detail::kernel& cl::sycl::detail::task::get_kernel ( )  [inline]
```

Get the kernel running if any.

Todo Specify this error in the spec

Definition at line 263 of file task.hpp.

References kernel.

Referenced by set_arg().

```
00263                                     {
00264     if (!kernel)
00265       throw non_cl_error("Cannot use an OpenCL kernel in this context");
00266     return *kernel;
00267   }
```

Here is the caller graph for this function:

10.27.3.6 get_queue()

```
auto cl::sycl::detail::task::get_queue ( ) [inline]
```

Get the queue behind the task to run a kernel on.

Definition at line 248 of file task.hpp.

References owner_queue.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::copy_in_cl_buffer(), cl::sycl::detail↩::accessor< T, Dimensions, Mode, Target >::get_cl_buffer(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor(), and cl::sycl::drt::set_kernel().

```
00248                    {
00249     return owner_queue;
00250   }
```

Here is the caller graph for this function:

10.27.3.7 notify_consumers()

```
void cl::sycl::detail::task::notify_consumers ( ) [inline]
```

Notify the waiting tasks that we are done.

Definition at line 153 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
00153                           {
00154     TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
00155    {
00156      std::unique_lock<std::mutex> ul { ready_mutex };
00157      execution_ended = true;
00158    }
00159     /* \todo Verify that the memory model with the notify does not
00160        require some fence or atomic */
00161     ready.notify_all();
00162   }
```

Here is the caller graph for this function:

10.27.3.8 postlude()

```
void cl::sycl::detail::task::postlude ( )  [inline]
```

Execute the epilogues.

Definition at line 224 of file task.hpp.

References p.

Referenced by schedule().

```
00224                   {
00225     for (const auto &p : epilogues)
00226       p();
00227     /* Free the functors that may own an accessor owning a buffer
00228        preventing the command group to complete */
00229     epilogues.clear();
00230   }
```

Here is the caller graph for this function:

10.27.3.9 prelude()

```
void cl::sycl::detail::task::prelude ( ) [inline]
```

Execute the prologues.

Definition at line 212 of file task.hpp.

References p, and TRISYCL_DUMP_T.

Referenced by schedule().

```
00212                 {
00213     TRISYCL_DUMP_T("task::prelude");
00214
00215     for (const auto &p : prologues)
00216       p();
00217     /* Free the functors that may own an accessor owning a buffer
00218        preventing the command group to complete */
00219     prologues.clear();
00220   }
```

Here is the caller graph for this function:

10.27.3.10 register_accessor()

```
std::size_t cl::sycl::detail::task::register_accessor (
            std::weak_ptr< detail::accessor_base > a ) [inline]
```

Register an accessor and return its registration order.

I would prefer to name this method `register` but this is a C++ keyword...

Definition at line 308 of file task.hpp.

Referenced by cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::register_accessor().

```
00308                                                                {
00309   accessors.push_back(a);
00310   return accessors.size() - 1;
00311 }
```

Here is the caller graph for this function:

10.27.3.11 release_buffers()

```
void cl::sycl::detail::task::release_buffers ( )  [inline]
```

Release the buffers that have been used by this task.

Definition at line 144 of file task.hpp.

References b, and TRISYCL_DUMP_T.

Referenced by schedule().

```
00144                          {
00145     TRISYCL_DUMP_T("Task " << this << " releases the written buffers");
00146     for (auto b: buffers_in_use)
00147       b->release();
00148     buffers_in_use.clear();
00149   }
```

Here is the caller graph for this function:

10.27.3.12 schedule()

```
void cl::sycl::detail::task::schedule (
            std::function< void(void) > f )  [inline]
```

Add a new task to the task graph and schedule for execution.

Detach the thread since it will synchronize by its own means

Todo This is an issue if there is an exception in the kernel

Definition at line 85 of file task.hpp.

References notify_consumers(), postlude(), prelude(), release_buffers(), TRISYCL_DUMP_T, and wait_for_producers().

```
00085                                           {
00086     /* To keep a copy of the task shared_ptr after the end of the
00087        command group, capture it by copy in the following lambda. This
00088        should be easier in C++17 with move semantics on capture
00089     */
00090     auto task = shared_from_this();
00091     auto execution = [=] {
00092       // Wait for the required tasks to be ready
00093       task->wait_for_producers();
00094       task->prelude();
00095       TRISYCL_DUMP_T("Execute the kernel");
00096       // Execute the kernel
00097       f();
00098       task->postlude();
00099       // Release the buffers that have been written by this task
00100       task->release_buffers();
00101       // Notify the waiting tasks that we are done
00102       task->notify_consumers();
00103       // Notify the queue we are done
00104       owner_queue->kernel_end();
00105       TRISYCL_DUMP_T("Task thread exit");
00106     };
00107     /* Notify the queue that there is a kernel submitted to the
00108        queue. Do not do it in the task contructor so that we can deal
00109        with command group without kernel and if we put it inside the
00110        thread, the queue may have finished before the thread is
00111        scheduled */
00112     owner_queue->kernel_start();
00113     /* \todo it may be implementable with packaged_task that would
00114        deal with exceptions in kernels
00115     */
00116 #ifndef TRISYCL_NO_ASYNC
00117     /* If in asynchronous execution mode, execute the functor in a new
00118        thread */
00119     std::thread thread(execution);
00120     TRISYCL_DUMP_T("Task thread started");
00121     /** Detach the thread since it will synchronize by its own means
00122
00123         \todo This is an issue if there is an exception in the kernel
00124     */
00125     thread.detach();
00126 #else
00127     // Just a synchronous execution otherwise
00128     execution();
00129 #endif
00130   }
```

Here is the call graph for this function:

10.27.3.13 set_arg() [1/2]

```
void cl::sycl::detail::task::set_arg (
            int arg_index,
            std::size_t arg_size,
            const void * scalar_value )  [inline]
```

Set a kernel argument by address.

Definition at line 271 of file task.hpp.

References cl::sycl::detail::kernel::get_boost_compute(), and get_kernel().

Referenced by cl::sycl::drt::serialize_accessor_arg(), and cl::sycl::drt::serialize_arg().

```
00271                                                                                {
00272 #ifdef TRISYCL_OPENCL
00273     // Forward to the OpenCL kernel
00274     get_kernel().get_boost_compute().set_arg(arg_index,
00275                                               arg_size,
00276                                               scalar_value);
00277 #else
00278     throw non_cl_error("Not compiled with OpenCL support");
00279 #endif
00280   }
```

Here is the call graph for this function:

Here is the caller graph for this function:

10.27.3.14 set_arg() [2/2]

```
template<typename T >
void cl::sycl::detail::task::set_arg (
            int arg_index,
            const T & scalar_value )  [inline]
```

Set a kernel argument by value.

Definition at line 285 of file task.hpp.

References cl::sycl::detail::kernel::get_boost_compute(), and get_kernel().

```
00285                                                   {
00286 #ifdef TRISYCL_OPENCL
00287     // Forward to the OpenCL kernel
00288     get_kernel().get_boost_compute().set_arg(arg_index, scalar_value);
00289 #else
00290     throw non_cl_error("Not compiled with OpenCL support");
00291 #endif
00292   }
```

Here is the call graph for this function:

10.27.3.15 set_kernel()

```
void cl::sycl::detail::task::set_kernel (
            const std::shared_ptr< cl::sycl::detail::kernel > & k )  [inline]
```

Set the OpenCL-compatible kernel running this task if any.

Definition at line 254 of file task.hpp.

Referenced by cl::sycl::drt::set_kernel().

```
00254                                                                   {
00255     kernel = k;
00256   }
```

Here is the caller graph for this function:

10.27.3.16 wait()

```
void cl::sycl::detail::task::wait ( ) [inline]
```

Wait for this task to be ready.

This is to be called from another thread

Definition at line 169 of file task.hpp.

References execution_ended, and TRISYCL_DUMP_T.

```
00169                {
00170     TRISYCL_DUMP_T("The task wait for task " << this << " to end");
00171     std::unique_lock<std::mutex> ul { ready_mutex };
00172     ready.wait(ul, [&] { return execution_ended; });
00173   }
```

10.27.3.17 wait_for_producers()

```
void cl::sycl::detail::task::wait_for_producers ( ) [inline]
```

Wait for the required producer tasks to be ready.

Definition at line 134 of file task.hpp.

References TRISYCL_DUMP_T.

Referenced by schedule().

```
00134                            {
00135     TRISYCL_DUMP_T("Task " << this << " waits for the producer tasks");
00136     for (auto &t : producer_tasks)
00137       t->wait();
00138     // We can let the producers rest in peace
00139     producer_tasks.clear();
00140   }
```

Here is the caller graph for this function:

10.27.4 Member Data Documentation

10.27.4.1 accessors

```
std::vector<std::weak_ptr<detail::accessor_base> > cl::sycl::detail::task::accessors
```

The accessors indexed by their creation order.

This is used to relate a kernel parameter of a kernel generated by the device compiler to its accessor.

Definition at line 76 of file task.hpp.

10.27.4.2 buffers_in_use

```
std::vector<std::shared_ptr<detail::buffer_base> > cl::sycl::detail::task::buffers_in_use
```

List of the buffers used by this task.

Todo Use a set to check that some buffers are not used many times at least on writing

Definition at line 45 of file task.hpp.

10.27.4.3 epilogues

```
std::vector<std::function<void(void)> > cl::sycl::detail::task::epilogues
```

Keep track of any epilogue to be executed after the kernel.

Definition at line 54 of file task.hpp.

10.27.4.4 execution_ended

```
bool cl::sycl::detail::task::execution_ended = false
```

Store if the execution ended, to be notified by task_ready.

Definition at line 57 of file task.hpp.

Referenced by wait().

10.27.4.5 kernel

`std::shared_ptr<detail::kernel> cl::sycl::detail::task::kernel`

The OpenCL-compatible kernel run by this task, if any.

Definition at line 70 of file task.hpp.

Referenced by get_kernel().

10.27.4.6 owner_queue

`std::shared_ptr<detail::queue> cl::sycl::detail::task::owner_queue`

Keep track of the queue used to submission to notify kernel completion or to run OpenCL kernels on.

Definition at line 67 of file task.hpp.

Referenced by get_queue().

10.27.4.7 producer_tasks

`std::vector<std::shared_ptr<detail::task> > cl::sycl::detail::task::producer_tasks`

The tasks producing the buffers used by this task.

Definition at line 48 of file task.hpp.

10.27.4.8 prologues

`std::vector<std::function<void(void)> > cl::sycl::detail::task::prologues`

Keep track of any prologue to be executed before the kernel.

Definition at line 51 of file task.hpp.

10.27.4.9 ready

`std::condition_variable cl::sycl::detail::task::ready`

To signal when this task is ready.

Definition at line 60 of file task.hpp.

10.27.4.10 ready_mutex

`std::mutex cl::sycl::detail::task::ready_mutex`

To protect the access to the condition variable.

Definition at line 63 of file task.hpp.

The documentation for this struct was generated from the following file:

- include/CL/sycl/command_group/detail/task.hpp

Chapter 11

File Documentation

11.1 include/CL/sycl.hpp File Reference

```
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/allocator.hpp"
#include "CL/sycl/address_space.hpp"
#include "CL/sycl/buffer.hpp"
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_runtime.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/error_handler.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/group.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/image.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/math.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/opencl_types.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/pipe.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"
#include "CL/sycl/static_pipe.hpp"
#include "CL/sycl/vec.hpp"
#include "CL/sycl/device_selector/detail/device_selector_tail.hpp"
#include "CL/sycl/device/detail/device_tail.hpp"
#include "CL/sycl/platform/detail/host_platform_tail.hpp"
#include "CL/sycl/platform/detail/opencl_platform_tail.hpp"
#include "CL/sycl/vendor/Xilinx/ssdm_inst.hpp"
#include "CL/sycl/vendor/Xilinx/opt_decorate_func.hpp"
```

```
#include "CL/sycl/vendor/Xilinx/partition_array.hpp"
```

Include dependency graph for sycl.hpp:

11.2 sycl.hpp

```
00001 /** \file
00002
00003     \mainpage
00004
00005     This is the main OpenCL SYCL C++ header file to experiment with
00006     the OpenCL CL provisional specification.
00007
00008     For more information about OpenCL SYCL:
00009     http://www.khronos.org/sycl/
00010
00011     For more information on this project and to access to the source of
00012     this file, look at https://github.com/triSYCL/triSYCL
00013
00014     The Doxygen version of the implementation itself is in
00015     http://Xilinx.github.io/triSYCL/Doxygen/triSYCL/html and
00016     http://Xilinx.github.io/triSYCL/Doxygen/triSYCL/triSYCL-implementation-refman.pdf
00017
00018
00019     Ronan at keryell dot FR
00020
00021     Copyright 2014--2015 Advanced Micro Devices, Inc.
00022
00023     Copyright 2015--2017 Xilinx, Inc.
00024
00025     This file is distributed under the University of Illinois Open Source
00026     License. See LICENSE.TXT for details.
00027 */
00028
00029
00030 /** Some global triSYCL configuration */
00031 #include "CL/sycl/detail/global_config.hpp"
00032 #include "CL/sycl/detail/default_classes.hpp"
00033
00034
00035 /* All the SYCL components, one per file */
00036 #include "CL/sycl/access.hpp"
00037 #include "CL/sycl/accessor.hpp"
00038 #include "CL/sycl/allocator.hpp"
00039 #include "CL/sycl/address_space.hpp"
00040 #include "CL/sycl/buffer.hpp"
00041 #include "CL/sycl/context.hpp"
00042 #include "CL/sycl/device.hpp"
00043 #include "CL/sycl/device_runtime.hpp"
00044 #include "CL/sycl/device_selector.hpp"
00045 #include "CL/sycl/error_handler.hpp"
00046 #include "CL/sycl/event.hpp"
00047 #include "CL/sycl/exception.hpp"
00048 #include "CL/sycl/group.hpp"
00049 #include "CL/sycl/handler.hpp"
00050 #include "CL/sycl/id.hpp"
00051 #include "CL/sycl/image.hpp"
00052 #include "CL/sycl/item.hpp"
00053 #include "CL/sycl/math.hpp"
00054 #include "CL/sycl/nd_item.hpp"
00055 #include "CL/sycl/nd_range.hpp"
00056 #include "CL/sycl/opencl_types.hpp"
00057 #include "CL/sycl/parallelism.hpp"
00058 #include "CL/sycl/pipe.hpp"
00059 #include "CL/sycl/pipe_reservation.hpp"
00060 #include "CL/sycl/platform.hpp"
00061 #include "CL/sycl/queue.hpp"
00062 #include "CL/sycl/range.hpp"
00063 #include "CL/sycl/static_pipe.hpp"
00064 #include "CL/sycl/vec.hpp"
```

```

// Some includes at the end to break some dependencies
#include "CL/sycl/device_selector/detail/device_selector_tail.hpp"
#include "CL/sycl/device/detail/device_tail.hpp"
#include "CL/sycl/platform/detail/host_platform_tail.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/platform/detail/opencl_platform_tail.hpp"
#endif

// Some include files for Xilinx-specific features.
// SSDM functions needs to be declared before their implementation
#include "CL/sycl/vendor/Xilinx/ssdm_inst.hpp"
#include "CL/sycl/vendor/Xilinx/opt_decorate_func.hpp"
#include "CL/sycl/vendor/Xilinx/partition_array.hpp"

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/
```

11.3 include/CL/sycl/access.hpp File Reference

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::access

  Describe the type of access by kernels.

Enumerations

- enum cl::sycl::access::mode {
  cl::sycl::access::mode::read = 42, cl::sycl::access::mode::write, cl::sycl::access::mode::read_write, cl::sycl↩
  ::access::mode::discard_write,
  cl::sycl::access::mode::discard_read_write, cl::sycl::access::mode::atomic }

This describes the type of the access mode to be used via accessor.

- enum cl::sycl::access::target { cl::sycl::access::target::global_buffer = 2014, cl::sycl::access::target::constant_buffer, cl::sycl::access::target::local, cl::sycl::access::target::image, cl::sycl::access::target::host_buffer, cl::sycl::access::target::host_image, cl::sycl::access::target::image_array, cl::sycl::access::target::pipe, cl::sycl::access::target::blocking_pipe }

  The target enumeration describes the type of object to be accessed via the accessor.

- enum cl::sycl::access::fence_space : char { cl::sycl::access::fence_space::local_space, cl::sycl::access::fence_space::global_space, cl::sycl::access::fence_space::global_and_local }

  Precise the address space a barrier needs to act on.

11.4 access.hpp

```
#ifndef TRISYCL_SYCL_ACCESS_HPP
#define TRISYCL_SYCL_ACCESS_HPP

/** \file The OpenCL SYCL access naming space

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

// SYCL dwells in the cl::sycl namespace
namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL

    @{
*/

/** Describe the type of access by kernels.

    \todo This values should be normalized to allow separate compilation
    with different implementations?
*/
namespace access {
  /* By using "enum mode" here instead of "enum struct mode", we have for
     example "write" appearing both as cl::sycl::access::mode::write and
     cl::sycl::access::write, instead of only the last one. This seems
     more conform to the specification. */

  /// This describes the type of the access mode to be used via accessor
  enum class mode {
    read = 42, /**< Read-only access. Insist on the fact that
                    read_write != read + write */
    write, ///< Write-only access, but previous content *not* discarded
    read_write, ///< Read and write access
    discard_write, ///< Write-only access and previous content discarded
    discard_read_write, /**< Read and write access and previous
                             content discarded*/
    atomic ///< Atomic access
  };


  /** The target enumeration describes the type of object to be accessed
      via the accessor
  */
  enum class target {
    global_buffer = 2014, //< Just pick a random number...
    constant_buffer,
    local,
    image,
    host_buffer,
    host_image,
    image_array,
    pipe,
    blocking_pipe
  };


  /** Precise the address space a barrier needs to act on
  */
```

```
  enum class fence_space : char {
    local_space,
    global_space,
    global_and_local
  };

}

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ACCESS_HPP
```

11.5 include/CL/sycl/accessor.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/local_accessor.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/detail/container_element_aspect.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/pipe/detail/pipe_accessor.hpp"
```

Include dependency graph for accessor.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::buffer< T, Dimensions, Allocator >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- class cl::sycl::pipe< T >

  A SYCL pipe. More...
- class cl::sycl::accessor< DataType, Dimensions, AccessMode, Target >

  The accessor abstracts the way buffer or pipe data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...
- class cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >

  The pipe accessor abstracts the way pipe data are accessed inside a kernel. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- template<typename Accessor >
  static auto & cl::sycl::get_pipe_detail (Accessor &a)

  Top-level function to break circular dependencies on the the types to get the pipe implementation.

11.6 accessor.hpp

```
#ifndef TRISYCL_SYCL_ACCESSOR_HPP
#define TRISYCL_SYCL_ACCESSOR_HPP

/** \file The OpenCL SYCL accessor<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/local_accessor.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/detail/container_element_aspect.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/pipe_reservation.hpp"
#include "CL/sycl/pipe/detail/pipe_accessor.hpp"

namespace cl {
namespace sycl {

template <typename T, int Dimensions, typename Allocator>
class buffer;
template <typename T>
class pipe;
class handler;

```

```
/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The accessor abstracts the way buffer or pipe data are accessed
    inside a kernel in a multidimensional variable length array way.

    \todo Implement it for images according so section 3.3.4.5
*/
template <typename DataType,
          int Dimensions,
          access::mode AccessMode,
          access::target Target = access::target::global_buffer>
class accessor :
    public detail::shared_ptr_implementation<accessor<DataType,
                                                      Dimensions,
                                                      AccessMode,
                                                      Target>,
                                             detail::accessor<DataType,
                                                              Dimensions,
                                                              AccessMode,
                                                              Target>>,
    public detail::container_element_aspect<DataType> {

 public:

  /// \todo in the specification: store the dimension for user request
  static constexpr auto dimensionality = Dimensions;

 private:

  using accessor_detail = typename detail::accessor<DataType,
                                                    Dimensions,
                                                    AccessMode,
                                                    Target>;

  // The type encapsulating the implementation
  using implementation_t = typename accessor::shared_ptr_implementation;

  // Allows the comparison operation to access the implementation
  friend implementation_t;

 public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** Construct a buffer accessor from a buffer using a command group
      handler object from the command group scope

      Constructor only available for global_buffer or constant_buffer
      target.

      access_target defines the form of access being obtained.

      \todo Add template allocator type in all the accessor
      constructors in the specification or just use a more opaque
      Buffer type?

      \todo fix specification where access mode should be target
      instead
  */
  template <typename Allocator>
  accessor(buffer<DataType, Dimensions, Allocator> & target_buffer,
           handler &command_group_handler) : implementation_t {
    new detail::accessor<DataType, Dimensions, AccessMode, Target> {
      target_buffer.implementation->implementation, command_group_handler }
  } {
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
    // Now the implementation is created, register it
    implementation->register_accessor();
  }


  /** Construct a buffer accessor from a buffer

      Constructor only available for host_buffer target.

      access_target defines the form of access being obtained.
  */
  template <typename Allocator>
```

```
  accessor(buffer<DataType, Dimensions, Allocator> &target_buffer)
    : implementation_t {
    new detail::accessor<DataType, Dimensions, AccessMode, Target>{
      target_buffer.implementation->implementation }
  } {
    static_assert(Target == access::target::host_buffer,
                  "without a handler, access target should be host_buffer");
  }


  /** Construct a buffer accessor from a buffer given a specific range for
      access permissions and an offset that provides the starting point
      for the access range using a command group handler object from the
      command group scope

      This accessor limits the processing of the buffer to the [offset,
      offset+range[ for every dimension. Any other parts of the buffer
      will be unaffected.

      Constructor only available for access modes global_buffer, and
      constant_buffer (see Table "Buffer accessor constructors").
      access_target defines the form of access being obtained.

      This accessor is recommended for discard-write and discard read
      write access modes, when the unaffected parts of the processing
      should be retained.
  */
  template <typename Allocator>
  accessor(buffer<DataType, Dimensions, Allocator> &target_buffer,
           handler &command_group_handler,
           const range<Dimensions> &offset,
           const range<Dimensions> &range) {
    detail::unimplemented();
  }


  /** Construct an accessor of dimension Dimensions with elements of type
      DataType using the passed range to specify the size in each
      dimension

      It needs as a parameter a command group handler object from the
      command group scope. Constructor only available if AccessMode is
      local, see Table 3.25.
  */
  accessor(const range<Dimensions> &allocation_size,
           handler &command_group_handler)
    : implementation_t { new detail::accessor<DataType,
                                              Dimensions,
                                              AccessMode,
                                              access::target::local> {
      allocation_size, command_group_handler
        }
  }
  {
    static_assert(Target == access::target::local,
                  "This accessor constructor requires "
                  "access target be local");
  }


  /** Return a range object representing the size of the buffer in
      terms of number of elements in each dimension as passed to the
      constructor

      \todo Move on
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
  */
  auto get_range() const {
    /* Interpret the shape which is a pointer to the first element as an
       array of Dimensions elements so that the range<Dimensions>
       constructor is happy with this collection

       \todo Add also a constructor in range<> to accept a const
       std::size_t *?
    */
    return implementation->get_range();
  }


  /** Returns the total number of elements behind the accessor

      Equal to get_range()[0] * ... * get_range()[Dimensions-1].

```

```
      \todo Move on
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
  */
  auto get_count() const {
    return implementation->get_count();
  }


  /** Returns the size of the underlying buffer storage in bytes

      \todo It is incompatible with buffer get_size() in the spec

      \todo Move on
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
  */
  auto get_size() const {
    return implementation->get_size();
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  typename accessor_detail::reference operator[](std::size_t index) {
//#ifdef TRISYCL_DEVICE
//#else
    return (*implementation)[index];
//#endif
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  typename accessor_detail::reference operator[](std::size_t index) const {
    return (*implementation)[index];
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) {
    return (*implementation)[index];
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) const {
    return (*implementation)[index];
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) {
    return (*this)[index.get_id()];
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) const {
    return (*this)[index.get_id()];
  }


  /** To use an accessor with an [nd_item<>]

      \todo Add in the specification because used by HPC-GPU slide 22
  */
  auto &operator[](nd_item<dimensionality> index) {
    return (*this)[index.get_global()];
  }

  /** To use an accessor with an [nd_item<>]

      \todo Add in the specification because used by HPC-GPU slide 22
  */
  auto &operator[](nd_item<dimensionality> index) const {
    return (*this)[index.get_global()];
  }


    /** Get the first element of the accessor
```

```

    Useful with an accessor on a scalar for example.

    \todo Add in the specification
*/
typename accessor_detail::reference operator*() {
  return **implementation;
}


/** Get the first element of the accessor

    Useful with an accessor on a scalar for example.

    \todo Add in the specification?

    \todo Add the concept of 0-dim buffer and accessor for scalar
    and use an implicit conversion to value_type reference to access
    the value with the accessor?
*/
typename accessor_detail::reference operator*() const {
  return **implementation;
}


/** Get the pointer to the start of the data

    \todo Should it be named data() instead? */
auto
get_pointer() const {
  return implementation->get_pointer();
}


/** Forward all the iterator functions to the implementation

    \todo Add these functions to the specification

    \todo The fact that the lambda capture make a const copy of the
    accessor is not yet elegantly managed... The issue is that
    begin()/end() dispatch is made according to the accessor
    constness and not from the array member constness...

    \todo try to solve it by using some enable_if on array
    constness?

    \todo The issue is that the end may not be known if it is
    implemented by a raw OpenCL cl_mem... So only provide on the
    device the iterators related to the start? Actually the accessor
    needs to know a part of the shape to have the multidimentional
    addressing. So this only require a size_t more...

    \todo Factor out these in a template helper
*/


// iterator begin() { return array.begin(); }
typename accessor_detail::iterator begin() const {
  return implementation->begin();
}


// iterator end() { return array.end(); }
typename accessor_detail::iterator end() const {
  return implementation->end();
}


// const_iterator begin() const { return implementation->begin(); }


// const_iterator end() const { return implementation->end(); }


typename accessor_detail::const_iterator cbegin() const {
  return implementation->cbegin();
}


typename accessor_detail::const_iterator cend() const {
  return implementation->cend();
}


typename accessor_detail::reverse_iterator rbegin() const {
  return implementation->rbegin();
};
```

```
  typename accessor_detail::reverse_iterator rend() const {
    return implementation->rend();
  }


  // const_reverse_iterator rbegin() const { return array.rbegin(); }


  // const_reverse_iterator rend() const { return array.rend(); }


  typename accessor_detail::const_reverse_iterator crbegin() const {
    return implementation->rbegin();
  }


  typename accessor_detail::const_reverse_iterator crend() const {
    return implementation->rend();
  }

};


/** The pipe accessor abstracts the way pipe data are accessed inside
    a kernel

    A specialization for an non-blocking pipe
*/
template <typename DataType,
          access::mode AccessMode>
class accessor<DataType, 1, AccessMode, access::target::pipe> :
    public detail::pipe_accessor<DataType, AccessMode, access::target::pipe> {
public:

  using accessor_detail =
    detail::pipe_accessor<DataType, AccessMode, access::target::pipe>;
  // Inherit of the constructors to have accessor constructor from detail
  using accessor_detail::accessor_detail;

  /** Construct a pipe accessor from a pipe using a command group
      handler object from the command group scope

      access_target defines the form of access being obtained.
  */
  accessor(pipe<DataType> &p, handler &command_group_handler)
    : accessor_detail { p.implementation, command_group_handler } { }

  /// Make a reservation inside the pipe
  pipe_reservation<accessor> reserve(std::size_t size) const {
    return accessor_detail::reserve(size);
  }


  /// Get the underlying pipe implementation
  auto &get_pipe_detail() {
    return accessor_detail::get_pipe_detail();
  }

};


/** The pipe accessor abstracts the way pipe data are accessed inside
    a kernel

    A specialization for a blocking pipe
*/
template <typename DataType,
          access::mode AccessMode>
class accessor<DataType, 1, AccessMode, access::target::blocking_pipe> :
    public detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>
 {
public:

  using accessor_detail =
    detail::pipe_accessor<DataType, AccessMode, access::target::blocking_pipe>;
  // Inherit of the constructors to have accessor constructor from detail
  using accessor_detail::accessor_detail;

  /** Construct a pipe accessor from a pipe using a command group
      handler object from the command group scope

      access_target defines the form of access being obtained.
  */
```

```
  accessor(pipe<DataType> &p, handler &command_group_handler)
    : accessor_detail { p.implementation, command_group_handler } { }


  /// Make a reservation inside the pipe
  pipe_reservation<accessor> reserve(std::size_t size) const {
    return accessor_detail::reserve(size);
  }


  /// Get the underlying pipe implementation
  auto &get_pipe_detail() {
    return accessor_detail::get_pipe_detail();
  }

};


/** Top-level function to break circular dependencies on the the types
    to get the pipe implementation */
template <typename Accessor>
static inline auto &get_pipe_detail(Accessor &a) {
  return a.get_pipe_detail();
  }

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ACCESSOR_HPP
```

11.7 include/CL/sycl/buffer/detail/accessor.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <boost/compute.hpp>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/accessor_base.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
```

Include dependency graph for accessor.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::buffer< T, Dimensions >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.8 accessor.hpp

```
#ifndef TRISYCL_SYCL_BUFFER_DETAIL_ACCESSOR_HPP
#define TRISYCL_SYCL_BUFFER_DETAIL_ACCESSOR_HPP

/** \file The OpenCL SYCL buffer accessor<> detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif
#include <boost/multi_array.hpp>

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor/detail/accessor_base.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
```

```
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"

namespace cl {
namespace sycl {

class handler;

namespace detail {

// Forward declaration of detail::buffer for use in accessor
template <typename T, int Dimensions> class buffer;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The buffer accessor abstracts the way buffer data are accessed
    inside a kernel in a multidimensional variable length array way.

    This implementation relies on boost::multi_array to provide this
    nice syntax and behaviour.

    Right now the aim of this class is just to access to the buffer in
    a read-write mode, even if capturing the multi_array_ref from a
    lambda make it const (since in examples we have lambda with [=]
    without mutable lambda).

    \todo Use the access::mode
*/
template <typename T,
          int Dimensions,
          access::mode Mode,
          access::target Target /* = access::global_buffer */>
class accessor :
    public detail::accessor_base,
    public std::enable_shared_from_this<accessor<T,
                                                 Dimensions,
                                                 Mode,
                                                 Target>>,
    public detail::debug<accessor<T,
                                  Dimensions,
                                  Mode,
                                  Target>> {
  /** Keep a reference to the accessed buffer

      Beware that it owns the buffer, which means that the accessor
      has to be destroyed to release the buffer and potentially
      unblock a kernel at the end of its execution
  */
  std::shared_ptr<detail::buffer<T, Dimensions>> buf;

  /// The implementation is a multi_array_ref wrapper
  using array_view_type = boost::multi_array_ref<T, Dimensions>;

  // The same type but writable
  using writable_array_view_type =
    typename std::remove_const<array_view_type>::type;

  /** The way the buffer is really accessed

      Use a mutable member because the accessor needs to be captured
      by value in the lambda which is then read-only. This is to avoid
      the user to use mutable lambda or have a lot of const_cast as
      previously done in this implementation
   */
  mutable array_view_type array;

public:

  /** \todo in the specification: store the dimension for user request

      \todo Use another name, such as from C++17 committee discussions.
   */
  static constexpr auto dimensionality = Dimensions;

  /** \todo in the specification: store the types for user request as STL
      or C++AMP */
  using value_type = T;
  using element = T;
  using reference = typename array_view_type::reference;
  using const_reference = typename array_view_type::const_reference;

  /** Inherit the iterator types from the implementation

      \todo Add iterators to accessors in the specification
  */
```

```
  using iterator = typename array_view_type::iterator;
  using const_iterator = typename array_view_type::const_iterator;
  using reverse_iterator = typename array_view_type::reverse_iterator;
  using const_reverse_iterator =
    typename array_view_type::const_reverse_iterator;


  /** Construct a host accessor from an existing buffer

      \todo fix the specification to rename target that shadows
      template parm
  */
  accessor(std::shared_ptr<detail::buffer<T, Dimensions>> target_buffer) :
    buf { target_buffer }, array { target_buffer->access } {
    target_buffer->template track_access_mode<Mode>();
    TRISYCL_DUMP_T("Create a host accessor write = " << is_write_access());
    static_assert(Target == access::target::host_buffer,
                  "without a handler, access target should be host_buffer");
    /* The host needs to wait for all the producers of the buffer to
       have finished */
    buf->wait();

#ifdef TRISYCL_OPENCL
    /* For the host context, we are obligated to update the buffer state
       during the accessors creation, otherwise we have no way of knowing
       if a buffer was modified on the host. This is only true because
       host accessors are blocking
     */
    cl::sycl::context ctx;
    buf->update_buffer_state(ctx, Mode, get_size(), array.data());
#endif
  }


  /** Construct a device accessor from an existing buffer

      \todo fix the specification to rename target that shadows
      template parm
  */
  accessor(std::shared_ptr<detail::buffer<T, Dimensions>> target_buffer,
           handler &command_group_handler) :
    buf { target_buffer }, array { target_buffer->access } {
    target_buffer->template track_access_mode<Mode>();
    TRISYCL_DUMP_T("Create a kernel accessor write = " << is_write_access());
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "access target should be global_buffer or constant_buffer "
                  "when a handler is used");
    // Register the buffer to the task dependencies
    task = buffer_add_to_task(buf, &command_group_handler, is_write_access());
  }


  /** Register the accessor once a \c std::shared_ptr is created on it

      This is to be called from outside once the object is created. It
      has been tried directly inside the contructor, but calling \c
      shared_from_this() from the constructor dead-lock with
      libstdc++6

      \todo Double-check with the C++ committee on this issue.
  */
  void register_accessor() {
    if (!task->get_queue()->is_host()) {
      // To keep alive this accessor in the following lambdas
      auto acc = this->shared_from_this();
      // Attach the accessor to the task and get its order
      set_order(task->register_accessor(acc));
#ifdef TRISYCL_OPENCL
      /* Before running the kernel, make sure the cl_mem behind this
         accessor is up-to-date on the device if needed and pass it to
         the kernel */
      task->add_prelude([=] {
          acc->copy_in_cl_buffer();
        });
      // After running the kernel, deal with some copy-back if needed
      task->add_postlude([=] {
          /* Even if this function does nothing, it is required to
             have the capture of acc to keep the accessor alive across
             the kernel execution up to the execution postlude */
          acc->copy_back_cl_buffer();
        });
```

```
#endif
    }
  }


  /** Return a range object representing the size of the buffer in
      terms of number of elements in each dimension as passed to the
      constructor

      \todo Move on
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
  */
  auto get_range() const {
    /* Interpret the shape which is a pointer to the first element as an
       array of Dimensions elements so that the range<Dimensions>
       constructor is happy with this collection

       \todo Add also a constructor in range<> to accept a const
       std::size_t *?
    */
    return range<Dimensions> {
      *(const std::size_t (*)[Dimensions])(array.shape())
        };
  }


  /** Returns the total number of elements behind the accessor

      Equal to get_range()[0] * ... * get_range()[Dimensions-1].

      \todo Move on
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
  */
  auto get_count() const {
    return array.num_elements();
  }


  /** Returns the size of the underlying buffer storage in bytes

      \todo Move on
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
  */
  auto get_size() const {
    return get_count()*sizeof(value_type);
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  reference operator[](std::size_t index) {
    return array[index];
  }


  /** Use the accessor with integers à la [][][]

      Use array_view_type::reference instead of auto& because it does not
      work in some dimensions.
   */
  reference operator[](std::size_t index) const {
    return array[index];
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) {
    return array(index);
  }


  /// To use the accessor with [id<>]
  auto &operator[](id<dimensionality> index) const {
    return array(index);
  }


  /// To use an accessor with [item<>]
  auto &operator[](item<dimensionality> index) {
    return (*this)[index.get()];
  }
```

```
/// To use an accessor with [item<>]
auto &operator[](item<dimensionality> index) const {
  return (*this)[index.get()];
}


/** To use an accessor with an [nd_item<>]

    \todo Add in the specification because used by HPC-GPU slide 22
*/
auto &operator[](nd_item<dimensionality> index) {
  return (*this)[index.get_global()];
}

/** To use an accessor with an [nd_item<>]

    \todo Add in the specification because used by HPC-GPU slide 22
*/
auto &operator[](nd_item<dimensionality> index) const {
  return (*this)[index.get_global()];
}


/** Get the first element of the accessor

    Useful with an accessor on a scalar for example.

    \todo Add in the specification
*/
reference operator*() {
  return *array.data();
}


/** Get the first element of the accessor

    Useful with an accessor on a scalar for example.

    \todo Add in the specification?

    \todo Add the concept of 0-dim buffer and accessor for scalar
    and use an implicit conversion to value_type reference to access
    the value with the accessor?
*/
reference operator*() const {
  return *array.data();
}


/// Get the buffer used to create the accessor
detail::buffer<T, Dimensions> &get_buffer() {
  return *buf;
}


/** Test if the accessor has a read access right

    \todo Strangely, it is not really constexpr because it is not a
    static method...

    \todo to move in the access::mode enum class and add to the
    specification ?
*/
constexpr bool is_read_access() const {
  return Mode == access::mode::read
    || Mode == access::mode::read_write
    || Mode == access::mode::discard_read_write;
}


/** Test if the accessor has a write access right

    \todo Strangely, it is not really constexpr because it is not a
    static method...

    \todo to move in the access::mode enum class and add to the
    specification ?
*/
constexpr bool is_write_access() const {
  return Mode == access::mode::write
    || Mode == access::mode::read_write
    || Mode == access::mode::discard_write
    || Mode == access::mode::discard_read_write;
}

```

```
  /** Return the pointer to the data

      \todo Implement the various pointer address spaces
  */
  auto
  get_pointer() const {
    return array.data();
  }


  /** Forward all the iterator functions to the implementation

      \todo Add these functions to the specification

      \todo The fact that the lambda capture make a const copy of the
      accessor is not yet elegantly managed... The issue is that
      begin()/end() dispatch is made according to the accessor
      constness and not from the array member constness...

      \todo try to solve it by using some enable_if on array
      constness?

      \todo The issue is that the end may not be known if it is
      implemented by a raw OpenCL cl_mem... So only provide on the
      device the iterators related to the start? Actually the accessor
      needs to know a part of the shape to have the multidimentional
      addressing. So this only require a size_t more...

      \todo Factor out these in a template helper

      \todo Do we need this in detail::accessor too or only in accessor?
  */


  // iterator begin() { return array.begin(); }
  iterator begin() const {
    return const_cast<writable_array_view_type &>(array).begin();
  }


  // iterator end() { return array.end(); }
  iterator end() const {
    return const_cast<writable_array_view_type &>(array).end();
  }


  // const_iterator begin() const { return array.begin(); }


  // const_iterator end() const { return array.end(); }


  const_iterator cbegin() const { return array.begin(); }


  const_iterator cend() const { return array.end(); }


  // reverse_iterator rbegin() { return array.rbegin(); }
  reverse_iterator rbegin() const {
    return const_cast<writable_array_view_type &>(array).rbegin();
  }


  // reverse_iterator rend() { return array.rend(); }
  reverse_iterator rend() const {
    return const_cast<writable_array_view_type &>(array).rend();
  }


  // const_reverse_iterator rbegin() const { return array.rbegin(); }


  // const_reverse_iterator rend() const { return array.rend(); }


  const_reverse_iterator crbegin() const { return array.rbegin(); }


  const_reverse_iterator crend() const { return array.rend(); }

private:
```

```
#ifdef TRISYCL_OPENCL
  // The following function are used from handler
  friend handler;

  /// Get the boost::compute::buffer or throw if unset
  boost::compute::buffer get_cl_buffer() const override {
    // This throws if not set
    auto ctx = task->get_queue()->get_context();
    return buf->get_cl_buffer(ctx);
  }


  /** Lazily associate a CL buffer to the SYCL buffer and copy data in it
      if required, updates the state of the data in the buffer across contexts
  */
  void copy_in_cl_buffer() {
    /* Create the OpenCL buffer and copy in it the data from the host if
       the buffer doesn't already exists or if the data is not up to date
    */
    auto ctx = task->get_queue()->get_context();
    buf->update_buffer_state(ctx, Mode, get_size(), array.data());
  }


  /// Does nothing
  void copy_back_cl_buffer() {
    /* The copy back is handled by the host accessor and the buffer destructor.
       We don't need to systematically transfer the data after the
       kernel execution

       \todo Figure out what to do with this function
    */
  }
#endif

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_ACCESSOR_HPP
```

11.9 include/CL/sycl/accessor/detail/accessor_base.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for accessor_base.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::accessor_base

  The buffer accessor base is here mainly to be accessed from the task without dependency on the buffer data types. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.10 accessor_base.hpp

```
#ifndef TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_BASE_HPP
#define TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_BASE_HPP

/** \file The OpenCL SYCL buffer accessor_base<> detail behind the
    scene to store the data-independent of the accessor

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"

namespace cl {
namespace sycl {
```

```
00025
00026 class handler;
00027
00028 namespace detail {
00029
00030 struct task;
00031
00032 /** \addtogroup data Data access and storage in SYCL
00033     @{
00034 */
00035
00036 /** The buffer accessor base is here mainly to be accessed from the
00037     task without dependency on the buffer data types
00038 */
00039 class accessor_base {
00040
00041 protected:
00042
00043   /// The task where the accessor is used in
00044   std::shared_ptr<detail::task> task;
00045
00046   /// The declaration order of the accessor
00047   std::size_t order;
00048
00049 public:
00050
00051   /// Set the order of the accessor
00052   void set_order(std::size_t o) {
00053     order = o;
00054   }
00055
00056
00057   /// Get the order of the accessor
00058   auto get_order() {
00059     return order;
00060   }
00061
00062
00063 #ifdef TRISYCL_OPENCL
00064   /// Get the boost::compute::buffer or throw if unset
00065   virtual boost::compute::buffer get_cl_buffer() const = 0;
00066 #endif
00067
00068 };
00069
00070 /// @} End the data Doxygen group
00071
00072 }
00073 }
00074 }
00075
00076 /*
00077     # Some Emacs stuff:
00078     ### Local Variables:
00079     ### ispell-local-dictionary: "american"
00080     ### eval: (flyspell-prog-mode)
00081     ### End:
00082 */
00083
00084 #endif // TRISYCL_SYCL_ACCESSOR_DETAIL_ACCESSOR_BASE_HPP
```

11.11 include/CL/sycl/accessor/detail/local_accessor.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <boost/compute.hpp>
#include <boost/multi_array.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
```

Include dependency graph for local_accessor.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...

- class cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >

  The local accessor specialization abstracts the way local memory is allocated to a kernel to be shared between work-items of the same work-group. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl
- cl::sycl::detail

11.12 local_accessor.hpp

```
#ifndef TRISYCL_SYCL_ACCESSOR_DETAIL_LOCAL_ACCESSOR_HPP
#define TRISYCL_SYCL_ACCESSOR_DETAIL_LOCAL_ACCESSOR_HPP

/** \file The OpenCL SYCL local accessor<> detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif
#include <boost/multi_array.hpp>

#include "CL/sycl/access.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"

namespace cl {
namespace sycl {

class handler;

namespace detail {

// Forward declaration of detail::accessor to declare the specialization
template <typename T,
          int Dimensions,
          access::mode Mode,
          access::target Target>
class accessor;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The local accessor specialization abstracts the way local memory
    is allocated to a kernel to be shared between work-items of the
    same work-group.

    \todo Use the access::mode
*/
template <typename T,
          int Dimensions,
          access::mode Mode>
class accessor<T, Dimensions, Mode, access::target::local> :
    public detail::debug<accessor<T,
                                  Dimensions,
                                  Mode,
                                  access::target::local>> {

  /// The implementation is a multi_array_ref wrapper
  using array_type = boost::multi_array_ref<T, Dimensions>;

  // The same type but writable
  // \todo Only if T is non const actually
  using writable_array_type =
    typename std::remove_const<array_type>::type;

  /** The way the buffer is really accessed

      Use a mutable member because the accessor needs to be captured
      by value in the lambda which is then read-only. This is to avoid
      the user to use mutable lambda or have a lot of const_cast as
      previously done in this implementation
   */
  mutable writable_array_type array;

  /** The allocation on the host for the local accessor

      Note that this is uninitialized memory, as stated in SYCL
      specification.
  */
  mutable T *allocation = nullptr;

public:
```

```

/** \todo in the specification: store the dimension for user request

    \todo Use another name, such as from C++17 committee discussions.
 */
static constexpr auto dimensionality = Dimensions;

/** \todo in the specification: store the types for user request as STL
    or C++AMP */
using value_type = T;
using element = T;
using reference = typename array_type::reference;
using const_reference = typename array_type::const_reference;

/** Inherit the iterator types from the implementation

    \todo Add iterators to accessors in the specification
*/
using iterator = typename array_type::iterator;
using const_iterator = typename array_type::const_iterator;
using reverse_iterator = typename array_type::reverse_iterator;
using const_reverse_iterator =
  typename array_type::const_reverse_iterator;


/** Construct a device accessor from an existing buffer

    \todo fix the specification to rename target that shadows
    template param
*/
accessor(const range<Dimensions> &allocation_size,
         handler &command_group_handler) :
  array { allocate_accessor(allocation_size) } {}


// Deallocate the memory
~accessor() {
  deallocate_accessor();
}


/** Return a range object representing the size of the buffer in
    terms of number of elements in each dimension as passed to the
    constructor

    \todo Move on
    https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
    https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
*/
auto get_range() const {
  /* Interpret the shape which is a pointer to the first element as an
     array of Dimensions elements so that the range<Dimensions>
     constructor is happy with this collection

     \todo Add also a constructor in range<> to accept a const
     std::size_t *?
  */
  return range<Dimensions> {
    *(const std::size_t (*)[Dimensions])(array.shape())
      };
}


/** Returns the total number of elements behind the accessor

    Equal to get_range()[0] * ... * get_range()[Dimensions-1].

    \todo Move on
    https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
    https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
*/
auto get_count() const {
  return array.num_elements();
}


/** Returns the size of the underlying buffer storage in bytes

    \todo Move on
    https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15564 and
    https://cvs.khronos.org/bugzilla/show_bug.cgi?id=14404
*/
auto get_size() const {
  return get_count()*sizeof(value_type);
}


```

```
/** Use the accessor with integers à la [][][]

    Use array_view_type::reference instead of auto& because it does not
    work in some dimensions.
 */
reference operator[](std::size_t index) {
  return array[index];
}


/** Use the accessor with integers à la [][][]

    Use array_view_type::reference instead of auto& because it does not
    work in some dimensions.
 */
reference operator[](std::size_t index) const {
  return array[index];
}


/// To use the accessor with [id<>]
auto &operator[](id<dimensionality> index) {
  return array(index);
}


/// To use the accessor with [id<>]
auto &operator[](id<dimensionality> index) const {
  return array(index);
}


/// To use an accessor with [item<>]
auto &operator[](item<dimensionality> index) {
  return (*this)[index.get()];
}


/// To use an accessor with [item<>]
auto &operator[](item<dimensionality> index) const {
  return (*this)[index.get()];
}


/** To use an accessor with an [nd_item<>]

    \todo Add in the specification because used by HPC-GPU slide 22
*/
auto &operator[](nd_item<dimensionality> index) {
  return (*this)[index.get_global()];
}

/** To use an accessor with an [nd_item<>]

    \todo Add in the specification because used by HPC-GPU slide 22
*/
auto &operator[](nd_item<dimensionality> index) const {
  return (*this)[index.get_global()];
}


/** Get the first element of the accessor

    Useful with an accessor on a scalar for example.

    \todo Add in the specification
*/
reference operator*() {
  return *array.data();
}


/** Get the first element of the accessor

    Useful with an accessor on a scalar for example.

    \todo Add in the specification?

    \todo Add the concept of 0-dim buffer and accessor for scalar
    and use an implicit conversion to value_type reference to access
    the value with the accessor?
*/
reference operator*() const {
  return *array.data();
}
```

```
  /** Test if the accessor has a read access right

      \todo Strangely, it is not really constexpr because it is not a
      static method...

      \todo to move in the access::mode enum class and add to the
      specification ?
  */
  constexpr bool is_read_access() const {
    return Mode == access::mode::read
      || Mode == access::mode::read_write
      || Mode == access::mode::discard_read_write;
  }


  /** Test if the accessor has a write access right

      \todo Strangely, it is not really constexpr because it is not a
      static method...

      \todo to move in the access::mode enum class and add to the
      specification ?
  */
  constexpr bool is_write_access() const {
    return Mode == access::mode::write
      || Mode == access::mode::read_write
      || Mode == access::mode::discard_write
      || Mode == access::mode::discard_read_write;
  }


  /** Forward all the iterator functions to the implementation

      \todo Add these functions to the specification

      \todo The fact that the lambda capture make a const copy of the
      accessor is not yet elegantly managed... The issue is that
      begin()/end() dispatch is made according to the accessor
      constness and not from the array member constness...

      \todo try to solve it by using some enable_if on array
      constness?

      \todo The issue is that the end may not be known if it is
      implemented by a raw OpenCL cl_mem... So only provide on the
      device the iterators related to the start? Actually the accessor
      needs to know a part of the shape to have the multidimentional
      addressing. So this only require a size_t more...

      \todo Factor out these in a template helper

      \todo Do we need this in detail::accessor too or only in accessor?
  */


  // iterator begin() { return array.begin(); }
  iterator begin() const {
    return const_cast<writable_array_type &>(array).begin();
  }


  // iterator end() { return array.end(); }
  iterator end() const {
    return const_cast<writable_array_type &>(array).end();
  }


  // const_iterator begin() const { return array.begin(); }


  // const_iterator end() const { return array.end(); }


  const_iterator cbegin() const { return array.begin(); }


  const_iterator cend() const { return array.end(); }


  // reverse_iterator rbegin() { return array.rbegin(); }
  reverse_iterator rbegin() const {
    return const_cast<writable_array_type &>(array).rbegin();
  }


```

```
  // reverse_iterator rend() { return array.rend(); }
  reverse_iterator rend() const {
    return const_cast<writable_array_type &>(array).rend();
  }


  // const_reverse_iterator rbegin() const { return array.rbegin(); }


  // const_reverse_iterator rend() const { return array.rend(); }


  const_reverse_iterator crbegin() const { return array.rbegin(); }


  const_reverse_iterator crend() const { return array.rend(); }

private:

  /// Allocate uninitialized buffer memory
  auto allocate_accessor(const range<Dimensions> &r) {
    auto count = r.get_count();
    // Allocate uninitialized memory
    allocation = std::allocator<value_type>{}.allocate(count);
    return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
  }


  /// Deallocate accessor memory
  void deallocate_accessor() {
    std::allocator<value_type>{}.deallocate(allocation, array.num_elements());
  }


  // The following function are used from handler
  friend handler;


};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ACCESSOR_DETAIL_LOCAL_ACCESSOR_HPP
```

11.13 include/CL/sycl/address_space/detail/address_space.hpp File Reference

Implement OpenCL address spaces in SYCL with C++-style.

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::ocl_type< T, AS >

  Generate a type with some real OpenCL 2 attribute if we are on an OpenCL device. More...
- struct cl::sycl::detail::ocl_type< T, constant_address_space >

  Add an attribute for __constant address space. More...
- struct cl::sycl::detail::ocl_type< T, global_address_space >

  Add an attribute for __global address space. More...
- struct cl::sycl::detail::ocl_type< T, local_address_space >

  Add an attribute for __local address space. More...
- struct cl::sycl::detail::ocl_type< T, private_address_space >

  Add an attribute for __private address space. More...
- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...
- struct cl::sycl::detail::address_space_base< T, AS >

  Implementation of the base infrastructure to wrap something in an OpenCL address space. More...
- struct cl::sycl::detail::address_space_variable< T, AS >

  Implementation of a variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_fundamental< T, AS >

  Implementation of a fundamental type with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_ptr< T, AS >

  Implementation for an OpenCL address space pointer. More...

- struct cl::sycl::detail::address_space_array< T, AS >

  Implementation of an array variable with an OpenCL address space. More...
- struct cl::sycl::detail::address_space_object< T, AS >

  Implementation of an object type with an OpenCL address space. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_GLOBAL_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(1))))

  Address space for OpenCL __global.
- #define TRISYCL_LOCAL_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(3))))

  Address space for OpenCL __local.
- #define TRISYCL_CONSTANT_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(2))))

  Address space for OpenCL __constant.
- #define TRISYCL_GENERIC_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(4))))

  Address space for OpenCL __generic.
- #define TRISYCL_PRIVATE_AS

  Address space for OpenCL __private.

Typedefs

- template<typename T , address_space AS>
  using cl::sycl::detail::addr_space = typename std::conditional< std::is_pointer< T >::value, address_space_ptr< T, AS >, typename std::conditional< std::is_class< T >::value, address_space_object< T, AS >, typename std::conditional< std::is_array< T >::value, address_space_array< T, AS >, address_space_fundamental< T, AS > >::type >::type >::type

  Dispatch the address space implementation according to the requested type.

11.13.1 Detailed Description

Implement OpenCL address spaces in SYCL with C++-style.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file address_space.hpp.

11.13.2 Macro Definition Documentation

11.13.2.1 TRISYCL_CONSTANT_AS

```
#define TRISYCL_CONSTANT_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(2))))
```

Address space for OpenCL __constant.

Definition at line 41 of file address_space.hpp.

11.13.2.2 TRISYCL_GENERIC_AS

```
#define TRISYCL_GENERIC_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(4))))
```

Address space for OpenCL __generic.

Definition at line 45 of file address_space.hpp.

11.13.2.3 TRISYCL_GLOBAL_AS

```
#define TRISYCL_GLOBAL_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(1))))
```

Address space for OpenCL __global.

Definition at line 35 of file address_space.hpp.

Referenced by cl::sycl::drt::accessor< Accessor >::accessor().

11.13.2.4 TRISYCL_LOCAL_AS

```
#define TRISYCL_LOCAL_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(3))))
```

Address space for OpenCL __local.

Definition at line 38 of file address_space.hpp.

11.13.2.5 TRISYCL_PRIVATE_AS

```
#define TRISYCL_PRIVATE_AS
```

Address space for OpenCL __private.

Definition at line 49 of file address_space.hpp.

11.14 address_space.hpp

```
#ifndef TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
#define TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP

/** \file

    Implement OpenCL address spaces in SYCL with C++-style.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/


/* Use macros for address spaces to be able to retarget more easily to
   different runtimes & compilers.

   For example can have OpenCL address spaces even if the address
   space parsing is off in Clang/LLVM. Look at SPIRAddrSpaceMap in
   tools/clang/lib/Basic/Targets.cpp:
   \code
     static const unsigned SPIRAddrSpaceMap[] = {
         1, // opencl_global
         3, // opencl_local
         2, // opencl_constant
         4, // opencl_generic
         0, // cuda_device
         0, // cuda_constant
         0  // cuda_shared
     };
   \endcode
*/

/// Address space for OpenCL __global
#define TRISYCL_GLOBAL_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(1))))

/// Address space for OpenCL __local
#define TRISYCL_LOCAL_AS TRISYCL_DEVICE_ONLY(__attribute__((address_space(3))))

/// Address space for OpenCL __constant
#define TRISYCL_CONSTANT_AS                                 \
  TRISYCL_DEVICE_ONLY(__attribute__((address_space(2))))

/// Address space for OpenCL __generic
#define TRISYCL_GENERIC_AS                                  \
  TRISYCL_DEVICE_ONLY(__attribute__((address_space(4))))

/// Address space for OpenCL __private
#define TRISYCL_PRIVATE_AS


namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup address_spaces
    @{
*/

/** Generate a type with some real OpenCL 2 attribute if we are on an
    OpenCL device

    In the general case, do not add any OpenCL address space qualifier */
template <typename T, address_space AS>
struct ocl_type { // NOTE: renamed from opencl_type because of MSVC bug
  using type = T;
};

/// Add an attribute for __constant address space
template <typename T>
struct ocl_type<T, constant_address_space> {
  using type = TRISYCL_CONSTANT_AS T;
};

#if TRISYCL_CL_LANGUAGE_VERSION >= 200
/// Add an attribute for __generic address space
template <typename T>
struct ocl_type<T, generic_address_space> {
  using type = TRISYCL_GENERIC_AS T;
};
#endif

/// Add an attribute for __global address space
template <typename T>
```

```
struct ocl_type<T, global_address_space> {
  using type = TRISYCL_GLOBAL_AS T;
};

/// Add an attribute for __local address space
template <typename T>
struct ocl_type<T, local_address_space> {
  using type = TRISYCL_LOCAL_AS T;
};

/// Add an attribute for __private address space
template <typename T>
struct ocl_type<T, private_address_space> {
  using type = TRISYCL_PRIVATE_AS T;
};


/* Forward declare some classes to allow some recursion in conversion
   operators */
template <typename SomeType, address_space SomeAS>
struct address_space_array;

template <typename SomeType, address_space SomeAS>
struct address_space_fundamental;

template <typename SomeType, address_space SomeAS>
struct address_space_object;

template <typename SomeType, address_space SomeAS>
struct address_space_ptr;

/** Dispatch the address space implementation according to the requested type

    \param T is the type of the object to be created

    \param AS is the address space to place the object into or to point to
    in the case of a pointer type
*/
template <typename T, address_space AS>
using addr_space =
  typename std::conditional<std::is_pointer<T>::value,
                            address_space_ptr<T, AS>,
  typename std::conditional<std::is_class<T>::value,
                            address_space_object<T, AS>,
  typename std::conditional<std::is_array<T>::value,
                            address_space_array<T, AS>,
                            address_space_fundamental<T, AS>
  >::type>::type>::type;


/** Implementation of the base infrastructure to wrap something in an
    OpenCL address space

    \param T is the type of the basic stuff to be created

    \param AS is the address space to place the object into

    \todo Verify/improve to deal with const/volatile?
*/
template <typename T, address_space AS>
struct address_space_base {
  /** Store the base type of the object

      \todo Add to the specification
  */
  using type = T;

  /** Store the base type of the object with OpenCL address space modifier

      \todo Add to the specification
  */
  using opencl_type = typename ocl_type<T, AS>::type;

  /** Set the address_space identifier that can be queried to know the
      pointer type */
  static auto constexpr address_space = AS;

};


/** Implementation of a variable with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into
*/
template <typename T, address_space AS>
```

```
struct address_space_variable : public address_space_base<T, AS> {
  /** Store the base type of the object with OpenCL address space modifier

      \todo Add to the specification
  */
  using opencl_type = typename ocl_type<T, AS>::type;

  /// Keep track of the base class as a short-cut
  using super = address_space_base<T, AS>;

protected:

  /* C++11 helps a lot to be able to have the same constructors as the
     parent class here

     \todo Add this to the list of required C++11 features needed for SYCL
  */
  opencl_type variable;

public:

  /** Allow to create an address space version of an object or to convert
      one to be used by the classes inheriting by this one because it is
      not possible to directly initialize a base class member in C++ */
  address_space_variable(const T & v) : variable(v) { }


  /// Put back the default constructors canceled by the previous definition
  address_space_variable() = default;


  /** Conversion operator to allow a address_space_object<T> to be used
      as a T so that all the methods of a T and the built-in operators for
      T can be used on a address_space_object<T> too.

      Use opencl_type so that if we take the address of it, the address
      space is kept.
  */
  operator opencl_type & () { return variable; }

  /// Return the address of the value to implement pointers
  opencl_type * get_address() { return &variable; }

};


/** Implementation of a fundamental type with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into

    \todo Verify/improve to deal with const/volatile?
*/
template <typename T, address_space AS>
struct address_space_fundamental : public
address_space_variable<T, AS> {
  /// Keep track of the base class as a short-cut
  using super = address_space_variable<T, AS>;

  /// Inherit from base class constructors
  using super::address_space_variable;


  /** Also request for the default constructors that have been disabled by
      the declaration of another constructor

      This ensures for example that we can write
      \code
        generic<float *> q;
      \endcode
      without initialization.
  */
  address_space_fundamental() = default;


  /** Allow for example assignment of a global<float> to a priv<double>
      for example

     Since it needs 2 implicit conversions, it does not work with the
     conversion operators already define, so add 1 more explicit
     conversion here so that the remaining implicit conversion can be
     found by the compiler.

     Strangely
     \code
     template <typename SomeType, address_space SomeAS>
```

```
     address_space_base(addr_space<SomeType, SomeAS>& v)
     : variable(SomeType(v)) { }
     \endcode
     cannot be used here because SomeType cannot be inferred. So use
     address_space_base<> instead

     Need to think further about it...
  */
  template <typename SomeType, cl::sycl::address_space SomeAS>
  address_space_fundamental(address_space_fundamental<SomeType, SomeAS>& v)
  {
    /* Strangely I cannot have it working in the initializer instead, for
       some cases */
    super::variable = SomeType(v);
  }

};


/** Implementation for an OpenCL address space pointer

    \param T is the pointer type

    Note that if \a T is not a pointer type, it is an error.

    All the address space pointers inherit from it, which makes trivial
    the implementation of \c cl::sycl::multi_ptr<T, AS>
*/
template <typename T, address_space AS>
struct address_space_ptr : public address_space_fundamental<T, AS> {
  // Verify that \a T is really a pointer
  static_assert(std::is_pointer<T>::value,
                "T must be a pointer type");

  /// Keep track of the base class as a short-cut
  using super = address_space_fundamental<T, AS>;

  /// Inherit from base class constructors
  using super::address_space_fundamental;

  using pointer_t = typename super::address_space_fundamental::type;
  using reference_t = typename std::remove_pointer_t<pointer_t>&;

  /** Allow initialization of a pointer type from the address of an
      element with the same type and address space
  */
  address_space_ptr(address_space_fundamental<typename std::pointer_traits<T>::element_type, AS> *p)
    : address_space_fundamental<T, AS> { p->get_address() } {}

  /// Put back the default constructors canceled by the previous definition
  address_space_ptr() = default;
};


/** Implementation of an array variable with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into
*/
template <typename T, address_space AS>
struct address_space_array : public address_space_variable<T, AS> {
  /// Keep track of the base class as a short-cut
  using super = address_space_variable<T, AS>;

  /// Inherit from base class constructors
  using super::address_space_variable;


  /** Allow to create an address space array from an array
   */
  address_space_array(const T &array) {
    std::copy(std::begin(array), std::end(array), std::begin(super::variable));
  };


  /** Allow to create an address space array from an initializer list

      \todo Extend to more than 1 dimension
  */
  address_space_array(std::initializer_list<std::remove_extent_t<T>> list) {
    std::copy(std::begin(list), std::end(list), std::begin(super::variable));
```

```
  };

};


/** Implementation of an object type with an OpenCL address space

    \param T is the type of the basic object to be created

    \param AS is the address space to place the object into

    The class implementation is just inheriting of T so that all methods
    and non-member operators on T work also on address_space_object<T>

    \todo Verify/improve to deal with const/volatile?

    \todo what about T having some final methods?
*/
template <typename T, address_space AS>
//struct address_space_object : public opencl_type<T, AS>::type,
struct address_space_object : public ocl_type<T, AS>::type,
                              public address_space_base<T, AS> {
  /** Store the base type of the object with OpenCL address space modifier

      \todo Add to the specification
  */
  using opencl_type = typename ocl_type<T, AS>::type;

  /* C++11 helps a lot to be able to have the same constructors as the
     parent class here but with an OpenCL address space

     \todo Add this to the list of required C++11 features needed for SYCL
  */
  using opencl_type::opencl_type;

  /** Allow to create an address space version of an object or to
      convert one */
  address_space_object(T && v) : opencl_type(v) { }

  /** Conversion operator to allow a address_space_object<T> to be used
      as a T so that all the methods of a T and the built-in operators for
      T can be used on a address_space_object<T> too.

      Use opencl_type so that if we take the address of it, the address
      space is kept. */
  operator opencl_type & () { return *this; }

};

/// @} End the address_spaces Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ADDRESS_SPACES_DETAIL_ADDRESS_SPACES_HPP
```

11.15 include/CL/sycl/address_space.hpp File Reference

Implement OpenCL address spaces in SYCL with C++-style.

```
#include "CL/sycl/address_space/detail/address_space.hpp"
```

Include dependency graph for address_space.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<typename T >
  using cl::sycl::constant = detail::addr_space< T, constant_address_space >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using cl::sycl::constant_ptr = constant< T * >

  Declare a variable to be in the OpenCL constant address space.
- template<typename T >
  using cl::sycl::generic = detail::addr_space< T, generic_address_space >

Declare a variable to be in the OpenCL 2 generic address space.

- template<typename T >
  using cl::sycl::global = detail::addr_space< T, global_address_space >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using cl::sycl::global_ptr = global< T * >

  Declare a variable to be in the OpenCL global address space.
- template<typename T >
  using cl::sycl::local = detail::addr_space< T, local_address_space >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using cl::sycl::local_ptr = local< T * >

  Declare a variable to be in the OpenCL local address space.
- template<typename T >
  using cl::sycl::priv = detail::addr_space< T, private_address_space >

  Declare a variable to be in the OpenCL private address space.
- template<typename T >
  using cl::sycl::private_ptr = priv< T * >

  Declare a variable to be in the OpenCL private address space.
- template<typename Pointer , address_space AS>
  using cl::sycl::multi_ptr = detail::address_space_ptr< Pointer, AS >

  A pointer that can be statically associated to any address-space.

Enumerations

- enum cl::sycl::address_space {
  cl::sycl::constant_address_space, cl::sycl::generic_address_space, cl::sycl::global_address_space, cl↩
  ::sycl::local_address_space,
  cl::sycl::private_address_space }

  Enumerate the different OpenCL 2 address spaces.

Functions

- template<typename T , address_space AS>
  multi_ptr< T, AS > cl::sycl::make_multi (multi_ptr< T, AS > pointer)

  Construct a cl::sycl::multi_ptr<> with the right type.

11.15.1 Detailed Description

Implement OpenCL address spaces in SYCL with C++-style.

Note that in SYCL 1.2, only pointer types should be specified but in this implementation we generalize the concept to any type.

Todo Add the alias ..._ptr<T> = ...<T *>

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file address_space.hpp.

11.16 address_space.hpp

```
#ifndef TRISYCL_SYCL_ADDRESS_SPACE_HPP
#define TRISYCL_SYCL_ADDRESS_SPACE_HPP

/** \file

    Implement OpenCL address spaces in SYCL with C++-style.

    Note that in SYCL 1.2, only pointer types should be specified but
    in this implementation we generalize the concept to any type.

    \todo Add the alias ..._ptr<T> = ...<T *>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup address_spaces Dealing with OpenCL address spaces
    @{
*/

/** Enumerate the different OpenCL 2 address spaces */
enum address_space {
  constant_address_space,
  generic_address_space,
  global_address_space,
  local_address_space,
  private_address_space,
};

}
}
/// @} End the address_spaces Doxygen group


#include "CL/sycl/address_space/detail/address_space.hpp"


namespace cl {
namespace sycl {

/** \addtogroup address_spaces
    @{
*/

/** Declare a variable to be in the OpenCL constant address space

    \param T is the type of the object
*/
template <typename T>
using constant = detail::addr_space<T, constant_address_space>
;


/** Declare a variable to be in the OpenCL constant address space

    \param T is the type of the object
*/
template <typename T>
using constant_ptr = constant<T*>;


/** Declare a variable to be in the OpenCL 2 generic address space

    \param T is the type of the object
*/
template <typename T>
using generic = detail::addr_space<T, generic_address_space>;


/** Declare a variable to be in the OpenCL global address space

    \param T is the type of the object
*/
template <typename T>
using global = detail::addr_space<T, global_address_space>
;


/** Declare a variable to be in the OpenCL global address space
```

```

    \param T is the type of the object
*/

template <typename T>
using global_ptr = global<T*>;


/** Declare a variable to be in the OpenCL local address space

    \param T is the type of the object
*/
template <typename T>
using local = detail::addr_space<T, local_address_space>;


/** Declare a variable to be in the OpenCL local address space

    \param T is the type of the object
*/
template <typename T>
using local_ptr = local<T*>;


/** Declare a variable to be in the OpenCL private address space

    \param T is the type of the object
*/
template <typename T>
using priv = detail::addr_space<T, private_address_space>;


/** Declare a variable to be in the OpenCL private address space

    \param T is the type of the object
*/
template <typename T>
using private_ptr = priv<T*>;


/** A pointer that can be statically associated to any address-space

    \param Pointer is the pointer type

    \param AS is the address space to point to

    Note that if \a Pointer is not a pointer type, it is an error.
*/
template <typename Pointer, address_space AS>
using multi_ptr = detail::address_space_ptr<Pointer, AS>;


/** Construct a cl::sycl::multi_ptr<> with the right type

    \param pointer is the address with its address space to point to

    \todo Implement the case with a plain pointer
*/
template <typename T, address_space AS>
multi_ptr<T, AS> make_multi(multi_ptr<T, AS> pointer) {
  return pointer;
}

}
}
/// @} End the parallelism Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ADDRESS_SPACE_HPP
```

11.17 include/CL/sycl/allocator.hpp File Reference

```
#include <memory>
```

Include dependency graph for allocator.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<typename T >
  using cl::sycl::buffer_allocator = std::allocator< T >

  The allocator objects give the programmer some control on how the memory is allocated inside SYCL.
- template<typename T >
  using cl::sycl::image_allocator = std::allocator< T >

  The allocator used for the `image` inside SYCL.
- template<typename T >
  using cl::sycl::map_allocator = std::allocator< T >

  The allocator used to map the memory at the same place.

11.18 allocator.hpp

```
#ifndef TRISYCL_SYCL_ALLOCATOR_HPP
#define TRISYCL_SYCL_ALLOCATOR_HPP

/** \file The OpenCL SYCL allocator

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The allocator objects give the programmer some control on how the
    memory is allocated inside SYCL
*/

/** The allocator used for the \c buffer inside SYCL

    Just use the default allocator for now.
*/
template <typename T>
using buffer_allocator = std::allocator<T>;


/** The allocator used for the \c image inside SYCL

    Just use the default allocator for now.
*/
template <typename T>
using image_allocator = std::allocator<T>;


/** The allocator used to map the memory at the same place

    Just use the default allocator for now.

    \todo : implement and clarify the specification. It looks like it
    is not really an allocator according the current spec
*/
template <typename T>
using map_allocator = std::allocator<T>;


/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ALLOCATOR_HPP
```

11.19 include/CL/sycl/buffer/detail/buffer.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <type_traits>
#include <boost/multi_array.hpp>
#include <boost/optional.hpp>
#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
```

```
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for buffer.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::buffer< T, Dimensions >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename BufferDetail >
  static std::shared_ptr< detail::task > cl::sycl::detail::buffer_add_to_task (BufferDetail buf, handler *command_group_handler, bool is_write_mode)

  Proxy function to avoid some circular type recursion.

11.20 buffer.hpp

```
#ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
#define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP

/** \file The OpenCL SYCL buffer<> detail implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>
#include <type_traits>

#include <boost/multi_array.hpp>
// \todo Use C++17 optional when it is mainstream
#include <boost/optional.hpp>

#include "CL/sycl/access.hpp"
#include "CL/sycl/buffer/detail/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {
namespace detail {


/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A SYCL buffer is a multidimensional variable length array (à la C99
    VLA or even Fortran before) that is used to store data to work on.

    In the case we initialize it from a pointer, for now we just wrap the
    data with boost::multi_array_ref to provide the VLA semantics without
    any storage.
*/
template <typename T,
          int Dimensions = 1>
class buffer : public detail::buffer_base,
               public detail::debug<buffer<T, Dimensions>> {
public:

  // Extension to SYCL: provide pieces of STL container interface
  using element = T;
  using value_type = T;
  /* Even if the buffer is read-only use a non-const type so at
     least the current implementation can copy the data too */
  using non_const_value_type = std::remove_const_t<value_type>;

private:

  // \todo Replace U and D somehow by T and Dimensions
  // To allow allocation access
  template <typename U,
            int D,
            access::mode Mode,
            access::target Target /* = access::global_buffer */>
    friend class detail::accessor;

  /** The allocator to be used when some memory is needed

      \todo Implement user-provided allocator
  */
  std::allocator<non_const_value_type> alloc;

```

```
  /** This is the multi-dimensional interface to the data that may point
      to either allocation in the case of storage managed by SYCL itself
      or to some other memory location in the case of host memory or
      storage<> abstraction use
  */
  boost::multi_array_ref<value_type, Dimensions> access;

  /** If some allocation is requested on the host for the buffer
      memory, this is where the memory is attached to.

      Note that this is uninitialized memory, as stated in SYCL
      specification.
  */
  non_const_value_type *allocation = nullptr;

  /* How to copy back data on buffer destruction, can be modified with
     set_final_data( ... )
   */
  boost::optional<std::function<void(void)>> final_write_back;

  // Keep the shared pointer used to create the buffer
  shared_ptr_class<T> input_shared_pointer;


  // Track if the buffer memory is provided as host memory
  bool data_host = false;

  // Track if data should be copied if a modification occurs
  bool copy_if_modified = false;

  // Track if data have been modified
  bool modified = false;

public:

  /// Create a new read-write buffer of size \param r
  buffer(const range<Dimensions> &r) : access { allocate_buffer(r) } {}


  /** Create a new read-write buffer from \param host_data of size
      \param r without further allocation */
  buffer(T *host_data, const range<Dimensions> &r) :
    access { host_data, r },
    data_host { true }
  {}


  /** Create a new read-only buffer from \param host_data of size \param r
      without further allocation

      If the buffer is non const, use a copy-on-write mechanism with
      internal writable memory.

      \todo Clarify the semantics in the spec. What happens if the
      host change the host_data after buffer creation?

      Only enable this constructor if the value type is not constant,
      because if it is constant, the buffer is constant too.
  */
  template <typename Dependent = T,
            typename = std::enable_if_t<!std::is_const<Dependent>::value>>
  buffer(const T *host_data, const range<Dimensions> &r) :
    /* The buffer is read-only, even if the internal multidimensional
       wrapper is not. If a write accessor is requested, there should
       be a copy on write. So this pointer should not be written and
       this const_cast should be acceptable. */
    access { const_cast<T *>(host_data), r },
    data_host { true },
    /* Set copy_if_modified to true, so that if an accessor with write
       access is created, data are copied before to be modified. */
    copy_if_modified { true }
  {}


  /** Create a new buffer with associated memory, using the data in
      host_data

      The ownership of the host_data is shared between the runtime and the
      user. In order to enable both the user application and the SYCL
      runtime to use the same pointer, a cl::sycl::mutex_class is
      used.
  */
  buffer(shared_ptr_class<T> &host_data, const range<Dimensions> &r) :
    access { host_data.get(), r },
    input_shared_pointer { host_data },
```

```
    data_host { true }
  {}


  /// Create a new allocated 1D buffer from the given elements
  template <typename Iterator>
  buffer(Iterator start_iterator, Iterator end_iterator) :
    access { allocate_buffer(std::distance(start_iterator, end_iterator)) }
    {
      /* Then assign allocation since this is the only multi_array
         method with this iterator interface */
      access.assign(start_iterator, end_iterator);
    }


  /** Create a new sub-buffer without allocation to have separate
      accessors later

      \todo To implement and deal with reference counting
  buffer(buffer<T, Dimensions> b,
         index<Dimensions> base_index,
         range<Dimensions> sub_range)
  */

  /// \todo Allow CLHPP objects too?
  ///
  /*
  buffer(cl_mem mem_object,
         queue from_queue,
         event available_event)
  */


  /** The buffer content may be copied back on destruction to some
      final location */
  ~buffer() {
#ifdef TRISYCL_OPENCL
    /* We ensure that the host has the most up-to-date version of the data
       before the buffer is destroyed. This is necessary because we do not
       systematically transfer the data back from a device with
       \c copy_back_cl_buffer any more.
       \todo Optimize for the case the buffer is not based on host memory
    */
    cl::sycl::context ctx;
    auto size = access.num_elements() * sizeof(value_type);
    call_update_buffer_state(ctx, access::mode::read, size,
access.data());

#endif
    if (modified && final_write_back)
      (*final_write_back)();
    // Allocate explicitly allocated memory if required
    deallocate_buffer();
  }


  /** Enforce the buffer to be considered as being modified.
      Same as creating an accessor with write access.
   */
  void mark_as_written() {
    modified = true;
  }


  /** This method is to be called whenever an accessor is created

      Its current purpose is to track if an accessor with write access
      is created and acting accordingly.
   */
  template <access::mode Mode,
            access::target Target = access::target::host_buffer>
  void track_access_mode() {
    // test if write access is required
    if (   Mode == access::mode::write
        || Mode == access::mode::read_write
        || Mode == access::mode::discard_write
        || Mode == access::mode::discard_read_write
        || Mode == access::mode::atomic
       ) {
      modified = true;
      if (copy_if_modified) {
        // Implement the allocate & copy-on-write optimization
        copy_if_modified = false;
        data_host = false;
        // Since \c allocate_buffer() changes \c access, keep a copy first
        auto current_access = access;
        /* The range is actually computed from \c access itself, so
```

```
          save it */
        auto current_range = get_range();
        allocate_buffer(current_range);
        /* Then move everything to the new place

          \todo Use std::uninitialized_move instead, when we switch
          to full C++17
        */
        std::copy(current_access.begin(),
                  current_access.end(),
                  access.begin());
      }
    }
  }


/** Return a range object representing the size of the buffer in
    terms of number of elements in each dimension as passed to the
    constructor
 */
 auto get_range() const {
   /* Interpret the shape which is a pointer to the first element as an
      array of Dimensions elements so that the range<Dimensions>
      constructor is happy with this collection

      \todo Add also a constructor in range<> to accept a const
      std::size_t *?
   */
   return range<Dimensions> {
     *(const std::size_t (*)[Dimensions])(access.shape())
       };
 }


 /** Returns the total number of elements in the buffer

     Equal to get_range()[0] * ... * get_range()[Dimensions-1].
 */
 auto get_count() const {
   return access.num_elements();
 }


 /** Returns the size of the buffer storage in bytes

     \todo rename to something else. In
     http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
     it is named bytes() for example
 */
 auto get_size() const {
   return get_count()*sizeof(value_type);
 }


 /** Set the weak pointer as destination for write-back on buffer
     destruction
 */
 void set_final_data(std::weak_ptr<T> && final_data) {
   final_write_back = [=] {
     if (auto sptr = final_data.lock()) {
       std::copy_n(access.data(), access.num_elements(), sptr.get());
     }
   };
 }


 /** Provide destination for write-back on buffer destruction as a
     shared pointer.
  */
 void set_final_data(std::shared_ptr<T> && final_data) {
   final_write_back = [=] {
     std::copy_n(access.data(), access.num_elements(), final_data.get());
   };
 }


 /** Disable write-back on buffer destruction as an iterator.
 */
 void set_final_data(std::nullptr_t) {
   final_write_back = boost::none;
 }


 /** Provide destination for write-back on buffer destruction as an
     iterator
 */
 template <typename Iterator>
```

```
  void set_final_data(Iterator final_data) {
 /*   using type_ = typename iterator_value_type<Iterator>::value_type;
    static_assert(std::is_same<type_, T>::value, "buffer type mismatch");
    static_assert(!(std::is_const<type_>::value),
                  "const iterator is not allowed");*/
    final_write_back = [=] {
      std::copy_n(access.data(), access.num_elements(), final_data);
    };
  }


private:

  /// Allocate uninitialized buffer memory
  auto allocate_buffer(const range<Dimensions> &r) {
    auto count = r.get_count();
    // Allocate uninitialized memory
    allocation = alloc.allocate(count);
    return boost::multi_array_ref<value_type, Dimensions> { allocation, r };
  }


  /// Deallocate buffer memory if required
  void deallocate_buffer() {
    if (allocation)
      alloc.deallocate(allocation, access.num_elements());
  }


  /** Function pair to work around the fact that T might be a \c const type.
      We call update_buffer_state only if T is not \c const, we have to
      use \c enable_if otherwise the compiler will try to cast \c const
      \c void* to \c void* if we create a buffer with a \c const type

      \todo Use \c if \c constexpr when it is available with C++17
  */
  template <typename BaseType = T, typename DataType>
  void call_update_buffer_state(cl::sycl::context ctx, access::mode mode,
                                size_t size, DataType* data,
                                std::enable_if_t<!std::is_const<BaseType>
                                ::value>* = 0) {
    update_buffer_state(ctx, mode, size, data);
  }


  /** Version of \c call_update_buffer_state that does nothing. It is called if
      the type of the data in the buffer is \c const
   */
  template <typename BaseType = T, typename DataType>
  void call_update_buffer_state(cl::sycl::context ctx, access::mode mode,
                                size_t size, DataType* data,
                                std::enable_if_t<std::is_const<BaseType>
                                ::value>* = 0) { }


public:

  /** Get a \c future to wait from inside the \c cl::sycl::buffer in
      case there is something to copy back to the host

      \return A \c future in the \c optional if there is something to
      wait for, otherwise an empty \c optional
      \todo Make the function private again
  */
  boost::optional<std::future<void>> get_destructor_future() {
    /* If there is only 1 shared_ptr user of the buffer, this is the
       caller of this function, the \c buffer_waiter, so there is no
       need to get a \ future otherwise there will be a dead-lock if
       there is only 1 thread waiting for itself.

       Since \c use_count() is applied to a \c shared_ptr just created
       for this purpose, it actually increase locally the count by 1,
       so check for 1 + 1 use count instead...
    */
    // If the buffer's destruction triggers a write-back, wait
    if ((shared_from_this().use_count() > 2) &&
        modified && (final_write_back || data_host)) {
      // Create a promise to wait for
      notify_buffer_destructor = std::promise<void> {};
      // And return the future to wait for it
      return notify_buffer_destructor->get_future();
    }
    return boost::none;
  }

```

```
private:

  // Allow buffer_waiter destructor to access get_destructor_future()
  // friend detail::buffer_waiter<T, Dimensions>::~buffer_waiter();
  /* \todo Work around to Clang bug
     https://llvm.org/bugs/show_bug.cgi?id=28873 cannot use destructor
     here */
  /* \todo solve the fact that get_destructor_future is not accessible
     when private and buffer_waiter uses a custom allocator */
  friend detail::buffer_waiter<T, Dimensions>;

};


/** Proxy function to avoid some circular type recursion

    \return a shared_ptr<task>

    \todo To remove with some refactoring
*/
template <typename BufferDetail>
static std::shared_ptr<detail::task>
buffer_add_to_task(BufferDetail buf,
                   handler *command_group_handler,
                   bool is_write_mode) {
    return buf->add_to_task(command_group_handler, is_write_mode);
  }

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_HPP
```

11.21 include/CL/sycl/buffer.hpp File Reference

```
#include <cstddef>
#include <iterator>
#include <memory>
#include <type_traits>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for buffer.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::buffer< T, Dimensions, Allocator >

  A SYCL buffer is a multidimensional variable length array (à la C99 VLA or even Fortran before) that is used to store data to work on. More...
- struct std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

11.22 buffer.hpp

```
#ifndef TRISYCL_SYCL_BUFFER_HPP
#define TRISYCL_SYCL_BUFFER_HPP

/** \file The OpenCL SYCL buffer<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <iterator>
#include <memory>
#include <type_traits>

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer/detail/buffer_waiter.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/event.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/queue.hpp"
#include "CL/sycl/range.hpp"

```

```
namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A SYCL buffer is a multidimensional variable length array (à la C99
    VLA or even Fortran before) that is used to store data to work on.

    \todo There is a naming inconsistency in the specification between
    buffer and accessor on T versus datatype

    \todo Finish allocator implementation

    \todo Think about the need of an allocator when constructing a buffer
    from other buffers

    \todo Update the specification to have a non-const allocator for
    const buffer? Or do we rely on rebind_alloc<T>. But does this work
    with astate-full allocator?

    \todo Add constructors from arrays so that in C++17 the range and
    type can be infered from the constructor

    \todo Add constructors from array_ref
*/
template <typename T,
          int Dimensions = 1,
          /* Even a buffer of const T may need to allocate memory, so
             need an allocator of non const T */
          typename Allocator = buffer_allocator<std::remove_const_t<T>>>
class buffer
  /* Use the underlying buffer waiter implementation that can be
     shared in the SYCL model */
  : public detail::shared_ptr_implementation<
                        buffer<T, Dimensions, Allocator>,
                        detail::buffer_waiter<T, Dimensions, Allocator>>,
    detail::debug<buffer<T, Dimensions, Allocator>> {
public:

  /// The STL-like types
  using value_type = T;
  using reference = value_type&;
  using const_reference = const value_type&;
  using allocator_type = Allocator;

private:

  // The type encapsulating the implementation
  using implementation_t = typename
buffer::shared_ptr_implementation;

  // Allows the comparison operation to access the implementation
  friend implementation_t;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** Use default constructors so that we can create a new buffer copy
      from another one, with either a l-value or an r-value (for
      std::move() for example).

      Since we just copy the shared_ptr<> from the
      shared_ptr_implementation above, this is where/how the sharing
      magic is happening with reference counting in this case.
  */
  buffer() = default;


  /** Create a new buffer of the given size with
      storage managed by the SYCL runtime

      The default behavior is to use the default host buffer
      allocator, in order to allow for host accesses. If the type of
      the buffer, has the const qualifier, then the default allocator
      will remove the qualifier to allow host access to the data.

      \param[in] r defines the size

      \param[in] allocator is to be used by the SYCL runtime
  */
  buffer(const range<Dimensions> &r, Allocator allocator = {})
    : implementation_t { detail::waiter<T, Dimensions, Allocator>(
                        new detail::buffer<T, Dimensions> { r }) }
```

```
    {}


/** Create a new buffer with associated host memory

    \param[in] host_data points to the storage and values used by
    the buffer

    \param[in] r defines the size

    \param[in] allocator is to be used by the SYCL runtime, of type
    \c cl::sycl::buffer_allocator<T> by default

    The host address is \code const T* \endcode, so the host memory
    is read-only.

    However, the typename T is not const so the device accesses can
    be both read and write accesses. Since, the host_data is const,
    this buffer is only initialized with this memory and there is
    no write after its destruction, unless there is another final
    data address given after construction of the buffer.

    Only enable this constructor if it is not the same as the one
    with \code const T *host_data \endcode, which is when \c T is
    already a constant type.

    \todo Actually this is redundant.
*/
template <typename Dependent = T,
          typename = std::enable_if_t<!std::is_const<Dependent>::value>>
buffer(const T *host_data,
       const range<Dimensions> &r,
       Allocator allocator = {})
  : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
                        { host_data, r }) }
{}


/** Create a new buffer with associated host memory

    \param[inout] host_data points to the storage and values used by
    the buffer

    \param[in] r defines the size

    \param[in] allocator is to be used by the SYCL runtime, of type
    cl::sycl::buffer_allocator<T> by default

    The memory is owned by the runtime during the lifetime of the
    object.  Data is copied back to the host unless the user
    overrides the behavior using the set_final_data method. host_data
    points to the storage and values used by the buffer and
    range<Dimensions> defines the size.
*/
buffer(T *host_data,
       const range<Dimensions> &r,
       Allocator allocator = {})
  : implementation_t { detail::waiter(new detail::buffer<T, Dimensions>
                        { host_data, r }) }
{}


/** Create a new buffer with associated memory, using the data in
    host_data

    \param[inout] host_data points to the storage and values used by
    the buffer

    \param[in] r defines the size

    \param[in] allocator is to be used by the SYCL runtime, of type
    cl::sycl::buffer_allocator<T> by default

    The ownership of the host_data is shared between the runtime and the
    user. In order to enable both the user application and the SYCL
    runtime to use the same pointer, a cl::sycl::mutex_class is
    used. The mutex m is locked by the runtime whenever the data is in
    use and unlocked otherwise. Data is synchronized with host_data, when
    the mutex is unlocked by the runtime.

    \todo update the specification to replace the pointer by a
    reference and provide the constructor with and without a mutex
*/
buffer(shared_ptr_class<T> &host_data,
       const range<Dimensions> &buffer_range,
```

```
        cl::sycl::mutex_class &m,
        Allocator allocator = {}) {
    detail::unimplemented();
  }


  /** Create a new buffer with associated memory, using the data in
      host_data

      \param[inout] host_data points to the storage and values used by
      the buffer

      \param[in] r defines the size

      \param[inout] m is the mutex used to protect the data access

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      The ownership of the host_data is shared between the runtime and the
      user. In order to enable both the user application and the SYCL
      runtime to use the same pointer, a cl::sycl::mutex_class is
      used.

      \todo add this mutex-less constructor to the specification
  */
  buffer(shared_ptr_class<T> host_data,
        const range<Dimensions> &buffer_range,
        Allocator allocator = {})
    : implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
                        { host_data, buffer_range }) }
  {}


  /** Create a new allocated 1D buffer initialized from the given
      elements ranging from first up to one before last

      The data is copied to an intermediate memory position by the
      runtime. Data is written back to the same iterator set if the
      iterator is not a const iterator.

      \param[inout] start_iterator points to the first element to copy

      \param[in] end_iterator points to just after the last element to copy

      \param[in] allocator is to be used by the SYCL runtime, of type
      cl::sycl::buffer_allocator<T> by default

      \todo Implement the copy back at buffer destruction

      \todo Generalize this for n-D and provide column-major and row-major
      initialization

      \todo a reason to have this nD is that
            set_final_data(weak_ptr_class<T> & finalData) is actually
            doing this linearization anyway

      \todo Allow read-only buffer construction too

      \todo update the specification to deal with forward iterators
      instead and rewrite back only when it is non const and output
      iterator at least

      \todo Allow initialization from ranges and collections à la STL
  */
  template <typename InputIterator,
            /* To force some iterator concept checking to avoid GCC 4.9
               diving into this when initializing from ({ int, int })
               which is a range<> and and not an iterator... */
            typename ValueType =
            typename std::iterator_traits<InputIterator>::value_type>
  buffer(InputIterator start_iterator,
        InputIterator end_iterator,
        Allocator allocator = {}) :
    implementation_t { detail::waiter(new
detail::buffer<T, Dimensions>
                        { start_iterator, end_iterator }) }
  {}


  /** Create a new sub-buffer without allocation to have separate
      accessors later

      \param[inout] b is the buffer with the real data

      \param[in] base_index specifies the origin of the sub-buffer inside the
```

```
      buffer b

      \param[in] sub_range specifies the size of the sub-buffer

      \todo To be implemented

      \todo Update the specification to replace index by id
  */
  buffer(buffer<T, Dimensions, Allocator> &b,
         const id<Dimensions> &base_index,
         const range<Dimensions> &sub_range,
         Allocator allocator = {}) { detail::unimplemented(); }


#ifdef TRISYCL_OPENCL
  /** Create a buffer from an existing OpenCL memory object associated
      with a context after waiting for an event signaling the
      availability of the OpenCL data

      \param[inout] mem_object is the OpenCL memory object to use

      \param[inout] from_queue is the queue associated to the memory
      object

      \param[in] available_event specifies the event to wait for if
      non null

      Note that a buffer created from a cl_mem object will only have
      one underlying cl_mem for the lifetime of the buffer and use on
      an incompatible queue constitues an error.

      \todo To be implemented

      \todo Improve the specification to allow CLHPP objects too
  */
  buffer(cl_mem mem_object,
         queue from_queue,
         event available_event = {},
         Allocator allocator = {}) { detail::unimplemented();  }
#endif


  // Use BOOST_DISABLE_ASSERTS at some time to disable range checking

  /** Get an accessor to the buffer with the required mode

      \param Mode is the requested access mode

      \param Target is the type of object to be accessed

      \param[in] command_group_handler is the command group handler in
      which the kernel is to be executed

      \todo Do we need for an accessor to increase the reference count of
      a buffer object? It does make more sense for a host-side accessor.

      \todo Implement the modes and targets
  */
  template <access::mode Mode,
            access::target Target = access::target::global_buffer>
  accessor<T, Dimensions, Mode, Target>
  get_access(handler &command_group_handler) {
    static_assert(Target == access::target::global_buffer
                  || Target == access::target::constant_buffer,
                  "get_access(handler) can only deal with access::global_buffer"
                  " or access::constant_buffer (for host_buffer accessor"
                  " do not use a command group handler");
    implementation->implementation->template track_access_mode<Mode, Target>();
    return { *this, command_group_handler };
  }


  /** Force the buffer to behave like if we had created
      an accessor in write mode.
   */
  void mark_as_written() {
    return implementation->implementation->mark_as_written();
  }


  /** Get a host accessor to the buffer with the required mode

      \param Mode is the requested access mode

      \todo Implement the modes
```

```

    \todo More elegant solution
*/
template <access::mode Mode,
          access::target Target = access::target::host_buffer>
accessor<T, Dimensions, Mode, Target>
get_access() {
  static_assert(Target == access::target::host_buffer,
                "get_access() without a command group handler is only"
                " for host_buffer accessor");
  implementation->implementation->template track_access_mode<Mode, Target>();
  return { *this };
}


/** Return a range object representing the size of the buffer in
    terms of number of elements in each dimension as passed to the
    constructor

    \todo rename to the equivalent from array_ref proposals? Such
    as size() in
    http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2016/p0009r2.html
*/
auto get_range() const {
  /* Interpret the shape which is a pointer to the first element as an
     array of Dimensions elements so that the range<Dimensions>
     constructor is happy with this collection
  */
  return implementation->implementation->get_range();
}


/** Returns the total number of elements in the buffer

    Equal to get_range()[0] * ... * get_range()[Dimensions-1].
*/
auto get_count() const {
  return implementation->implementation->get_count();
}


/** Returns the size of the buffer storage in bytes

    Equal to get_count()*sizeof(T).

    \todo rename to something else. In
    http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0122r0.pdf
    it is named bytes() for example
*/
size_t get_size() const {
  return implementation->implementation->get_size();
}


/** Returns the number of buffers that are shared/referenced

    For example
    \code
    cl::sycl::buffer<int> b { 1000 };
    // Here b.use_count() should return 1
    cl::sycl::buffer<int> c { b };
    // Here b.use_count() and b.use_count() should return 2
    \endcode

    \todo Add to the specification, useful for validation
*/
auto use_count() const {
  // Rely on the shared_ptr<> use_count()
  return implementation.use_count();
}


/** Ask for read-only status of the buffer

    \todo Add to specification
*/
bool constexpr is_read_only() const {
  return std::is_const<T>::value;
}


/** Set destination of buffer data on destruction

    The finalData points to the host memory to which, the outcome of all
    the buffer processing is going to be copied to.

    This is the final pointer, which is going to be accessible after the
```

```
      destruction of the buffer and in the case where this is a valid
      pointer, the data are going to be copied to this host address.

      finalData is different from the original host address, if the buffer
      was created associated with one. This is mainly to be used when a
      shared_ptr is given in the constructor and the output data will
      reside in a different location from the initialization data.

      It is defined as a weak_ptr referring to a shared_ptr that is not
      associated with the cl::sycl::buffer, and so the cl::sycl::buffer
      will have no ownership of finalData.

      \todo Update the API to take finalData by value instead of by
            reference.  This way we can have an implicit conversion
            possible at the API call from a shared_ptr<>, avoiding an
            explicit weak_ptr<> creation

      \todo figure out how set_final_data() interact with the other
      way to write back some data or with some data sharing with the
      host that can not be undone
  */
  void set_final_data(shared_ptr_class<T> finalData) {
    implementation->implementation->set_final_data(std::move(finalData));
  }


  /** Set destination of buffer data on destruction.
   */
  void set_final_data(weak_ptr_class<T> finalData) {
    implementation->implementation->set_final_data(std::move(finalData));
  }


  /** Disable write-back on buffer destruction.
   */
  void set_final_data(std::nullptr_t) {
    implementation->implementation->set_final_data(nullptr);
  }


#ifdef TRISYCL_OPENCL
  /** Check if the buffer is already cached in a certain context
   */
  bool is_cached(cl::sycl::context& ctx) {
    return implementation->implementation->is_cached(ctx);
  }


  /** Check if the data stored in the buffer is up-to-date in a certain context
   */
  bool is_data_up_to_date(cl::sycl::context& ctx) {
    return implementation->implementation->is_data_up_to_date(ctx);
  }
#endif

  /** Set destination of buffer data on destruction.

      WARNING: the user has to ensure that the object refered to by the
      iterator will be alive after buffer destruction, otherwise the behaviour
      is undefined.
   */
  template<typename Iterator>
  void set_final_data(Iterator&& finalData) {
    implementation->implementation->
      set_final_data(std::forward<Iterator>(finalData));
  }

};

/// @} End the data Doxygen group

}
}

/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <typename T,
          int Dimensions,
          typename Allocator>
struct hash<cl::sycl::buffer<T, Dimensions, Allocator>> {

  auto operator()(const cl::sycl::buffer<T, Dimensions, Allocator>
```

```
      &b) const {
00545     // Forward the hashing to the implementation
00546     return b.hash();
00547   }
00548
00549 };
00550
00551 }
00552
00553 /*
00554     # Some Emacs stuff:
00555     ### Local Variables:
00556     ### ispell-local-dictionary: "american"
00557     ### eval: (flyspell-prog-mode)
00558     ### End:
00559 */
00560
00561 #endif // TRISYCL_SYCL_BUFFER_HPP
```

11.23 include/CL/sycl/buffer/detail/buffer_base.hpp File Reference

```
#include <atomic>
#include <boost/compute.hpp>
#include <boost/optional.hpp>
#include <condition_variable>
#include <future>
#include <memory>
#include <mutex>
#include <unordered_set>
#include <utility>
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/context.hpp"
```

Include dependency graph for buffer_base.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::buffer_base

  Factorize some template independent buffer aspects in a base class. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- static std::shared_ptr< detail::task > cl::sycl::detail::add_buffer_to_task (handler *command_group_handler, std::shared_ptr< detail::buffer_base > b, bool is_write_mode)

11.24 buffer_base.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
00002 #define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
00003
00004 /** \file The buffer_base behind the buffers, independent of the data
00005     type
00006
00007     Ronan at Keryell point FR
00008
00009     This file is distributed under the University of Illinois Open Source
00010     License. See LICENSE.TXT for details.
```

```
*/

#include <atomic>
#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif
// \todo Use C++17 optional when it is mainstream
#include <boost/optional.hpp>
#include <condition_variable>
#include <future>
#include <memory>
#include <mutex>
#include <unordered_set>
#include <utility>

#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/context.hpp"

namespace cl {
namespace sycl {

class handler;

namespace detail {

struct task;
struct buffer_base;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

inline static std::shared_ptr<detail::task>
add_buffer_to_task(handler *command_group_handler,
                   std::shared_ptr<detail::buffer_base> b,
                   bool is_write_mode);

/** Factorize some template independent buffer aspects in a base class
 */
struct buffer_base : public std::enable_shared_from_this<buffer_base> {

  //// Keep track of the number of kernel accessors using this buffer
  std::atomic<size_t> number_of_users;

  /// Track the latest task to produce this buffer
  std::weak_ptr<detail::task> latest_producer;
  /// To protect the access to latest_producer
  std::mutex latest_producer_mutex;

  /// To signal when this buffer ready
  std::condition_variable ready;
  /// To protect the access to the condition variable
  std::mutex ready_mutex;

  /** If the SYCL user buffer destructor is blocking, use this to
      block until this buffer implementation is destroyed.

      Use a void promise since there is no value to send, only
      waiting */
  boost::optional<std::promise<void>> notify_buffer_destructor;

  /// To track contexts in which the data is up-to-date
  std::unordered_set<cl::sycl::context> fresh_ctx;

#ifdef TRISYCL_OPENCL
  /** Buffer-side cache that keeps the \c boost::compute::buffer (and the
      underlying \c cl_buffer ) so that if the buffer already exists inside
      the same context it is not recreated.
   */
  std::unordered_map<cl::sycl::context, boost::compute::buffer> buffer_cache;
#endif

  /** Create a buffer base and marks the host context as the context that
      holds the most recent version of the data
      \todo Use lazy allocation for the context tracking set
   */
  buffer_base() : number_of_users { 0 },
                  fresh_ctx { cl::sycl::context {} } {}


  /// The destructor wait for not being used anymore
  ~buffer_base() {
    wait();
    // If there is the last SYCL user buffer waiting, notify it
    if (notify_buffer_destructor)
      notify_buffer_destructor->set_value();
  }
```

```


  /// Wait for this buffer to be ready, which is no longer in use
  void wait() {
    std::unique_lock<std::mutex> ul { ready_mutex };
    ready.wait(ul, [&] {
        // When there is no producer for this buffer, we are ready to use it
        return number_of_users == 0;
      });
  }


  /// Mark this buffer in use by a task
  void use() {
    // Increment the use count
    ++number_of_users;
  }


  /// A task has released the buffer
  void release() {
    if (--number_of_users == 0)
      // Notify the host consumers or the buffer destructor that it is ready
      ready.notify_all();
  }


  /// Return the latest producer for the buffer
  std::shared_ptr<detail::task> get_latest_producer() {
    std::lock_guard<std::mutex> lg { latest_producer_mutex };
    // Return the valid shared_ptr to the task, if any
    return latest_producer.lock();
  }


  /** Return the latest producer for the buffer and set another
      future producer
  */
  std::shared_ptr<detail::task>
  set_latest_producer(std::weak_ptr<detail::task> newer_latest_producer) {
    std::lock_guard<std::mutex> lg { latest_producer_mutex };
    using std::swap;

    swap(newer_latest_producer, latest_producer);
    // Return the valid shared_ptr to the previous producing task, if any
    return newer_latest_producer.lock();
  }


  /// Add a buffer to the task running the command group
  std::shared_ptr<detail::task>
  add_to_task(handler *command_group_handler, bool is_write_mode) {
    return add_buffer_to_task(command_group_handler,
                              shared_from_this(),
                              is_write_mode);
  }


#ifdef TRISYCL_OPENCL
  /// Check if the data of this buffer is up-to-date in a certain context
  bool is_data_up_to_date(const cl::sycl::context& ctx) {
    return fresh_ctx.count(ctx);
  }


  /// Check if the buffer is already cached for a certain context
  bool is_cached(const cl::sycl::context& ctx) {
    return buffer_cache.count(ctx);
  }


  /** Create a \c boost::compute::buffer for this \c cl::sycl::buffer in the
      cache and associate it with a given context
  */
  void create_in_cache(const cl::sycl::context& ctx, size_t size,
                       cl_mem_flags flags, void* data) {
    buffer_cache[ctx] = boost::compute::buffer
      { ctx.get_boost_compute(),
        size,
        flags,
        data
      };
  }


  /** Transfer the most up-to-date version of the data to the host
      if the host version is not already up-to-date
```

```
*/
void sync_with_host(std::size_t size, void* data) {
  cl::sycl::context host_context;
  if (!is_data_up_to_date(host_context) && !fresh_ctx.empty()) {
    /* We know that the context(s) in \c fresh_ctx hold the most recent
       version of the buffer
    */
    auto fresh_context = *(fresh_ctx.begin());
    auto fresh_q = fresh_context.get_boost_queue();
    fresh_q.enqueue_read_buffer(buffer_cache[fresh_context], 0, size, data);
    fresh_ctx.insert(host_context);
  }
}


/** When a transfer is requested this function is called, it will
    update the state of the buffer according to the context in which
    the accessor is created and the access mode
*/
void update_buffer_state(const cl::sycl::context& target_ctx,
                         access::mode mode, std::size_t size, void* data) {
  /* The \c cl_buffer we put in the cache might get accessed again in the
     future, this means that we have to always to create it in read/write
     mode to be able to write to it if it is accessed through a
     write accessor in the future
   */
  auto constexpr flag = CL_MEM_READ_WRITE;

  /* The buffer is accessed in read mode, we want to transfer the data only if
     necessary. We start a transfer if the data on the target context is not
     up to date and then update the fresh context set.
   */
  if (mode == access::mode::read) {

    if (is_data_up_to_date(target_ctx))
      // If read mode and the data is up-to-date there is nothing to do
      return;

    // The data is not up-to-date, we need a transfer
    // We also want to be sure that the host holds the most recent data
    sync_with_host(size, data);

    if (!target_ctx.is_host()) {
      // If the target context is a device context
      if (!is_cached(target_ctx)) {
        /* If not cached, we create the buffer and copy the data
           at the same time
        */
        create_in_cache(target_ctx, size,
                        (flag | CL_MEM_COPY_HOST_PTR), data);
        fresh_ctx.insert(target_ctx);
        return;
      }

      /* Else we transfer the data to the existing buffer associated
         with the target context buffer
      */
      auto q = target_ctx.get_boost_queue();
      q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
      fresh_ctx.insert(target_ctx);
    }
    return;
  }

  /* The buffer might be written to, this means that we have to consider
     every version of the data obsolete except in the target context

     We go through the same process as in read mode but in addition
     we empty the fresh context set and just add the target context

     If the data is up to date on the target we just have to update
     the context set and nothing else
  */
  if (!is_data_up_to_date(target_ctx)) {

    if (   mode == access::mode::read_write
        || mode == access::mode::write
        || mode == access::mode::atomic) {
      // If the data is not up-to-date in the target context
      // We want to host to be up-to-date
      sync_with_host(size, data);

      if (!target_ctx.is_host()) {
        // If the target context is a device context
        if (!is_cached(target_ctx)) {
          create_in_cache(target_ctx, size,
                          (flag | CL_MEM_COPY_HOST_PTR), data);
```

```
          }
          else {
            // We update the buffer associated with the target context
            auto q = target_ctx.get_boost_queue();
            q.enqueue_write_buffer(buffer_cache[target_ctx], 0, size, data);
          }
        }
      }

      /* When in discard mode we don't need to transfer any data, we just create
         the \c cl_buffer if it doesn't exist in the cache
      */
      if (   mode == access::mode::discard_write
          || mode == access::mode::discard_read_write) {
        /* We only need to create the buffer if it doesn't exist
           but without copying any data because of the discard mode
        */
        if (!target_ctx.is_host() && !is_cached(target_ctx)) {
          // If the context doesn't exist we create it.
          /* We don't want to transfer any data so we don't
             add \c CL_MEM_COPY_HOST_PTR
          */
          create_in_cache(target_ctx, size, flag, 0);
        }
      }
    }
    /* Here we are sure that we are in some kind of write mode,
       we indicate that all contexts except the target context
       are not up-to-date anymore
    */
    fresh_ctx.clear();
    fresh_ctx.insert(target_ctx);
  }


  /// Returns the cl_buffer for a given context.
  boost::compute::buffer get_cl_buffer(const cl::sycl::context&
context) {
    return buffer_cache[context];
  }

#endif

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_BASE_HPP
```

11.25 include/CL/sycl/buffer/detail/buffer_waiter.hpp File Reference

```
#include <cstddef>
#include <future>
#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
```

Include dependency graph for buffer_waiter.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >

  A helper class to wait for the final buffer destruction if the conditions for blocking are met. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename T , int Dimensions = 1, typename Allocator = buffer_allocator<std::remove_const_t<T>>>
  auto cl::sycl::detail::waiter (detail::buffer< T, Dimensions > *b)

  Helper function to create a new buffer_waiter.

11.26 buffer_waiter.hpp

```
#ifndef TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP
#define TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP

/** \file A helper class to wait for the buffer<> detail

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <future>

#include "CL/sycl/buffer/detail/buffer.hpp"
#include "CL/sycl/buffer_allocator.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A helper class to wait for the final buffer destruction if the
    conditions for blocking are met
*/
template <typename T,
          int Dimensions = 1,
          typename Allocator = buffer_allocator<std::remove_const_t<T>>>
class buffer_waiter :
    public detail::shared_ptr_implementation<buffer_waiter<T,
                                                           Dimensions,
                                                           Allocator>,
                                             detail::buffer<T, Dimensions>>,
    detail::debug<buffer_waiter<T, Dimensions, Allocator>> {

  // The type encapsulating the implementation
  using implementation_t = typename buffer_waiter::shared_ptr_implementation;

  // Allows the comparison operation to access the implementation
  friend implementation_t;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /// Create a new buffer_waiter on top of a detail::buffer
  buffer_waiter(detail::buffer<T, Dimensions> *b) : implementation_t { b } {}


  /** The buffer_waiter destructor waits for any data to be written
      back to the host, if any
  */
  ~buffer_waiter() {
    /* Get a future from the implementation if we have to wait for its
       destruction */
    auto f = implementation->get_destructor_future();
    if (f) {
      /* No longer carry for the implementation buffer which is free to
         live its life up to its destruction */
      implementation.reset();
      TRISYCL_DUMP_T("~buffer_waiter() is waiting");
      // Then wait for its end in some other thread
      f->wait();
      TRISYCL_DUMP_T("~buffer_waiter() is done");
```

```
    }
  }
};


/// Helper function to create a new buffer_waiter
template <typename T,
          int Dimensions = 1,
          typename Allocator = buffer_allocator<std::remove_const_t<T>
>>
inline auto waiter(detail::buffer<T, Dimensions> *
b) {
  return new buffer_waiter<T, Dimensions, Allocator> { b };
}

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_DETAIL_BUFFER_WAITER_HPP
```

11.27 include/CL/sycl/buffer_allocator.hpp File Reference

```
#include <cstddef>
#include <memory>
```

Include dependency graph for buffer_allocator.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.28 buffer_allocator.hpp

```
00001 #ifndef TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
00002 #define TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
00003
00004 /** \file The OpenCL SYCL buffer_allocator
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013 #include <memory>
00014
00015 namespace cl {
00016 namespace sycl {
00017
00018 /** \addtogroup data Data access and storage in SYCL
```

```
    @{
*/

/** The default buffer allocator used by the runtime, when no allocator is
    defined by the user

    Reuse the C++ default allocator.
*/
template <typename T>
using buffer_allocator = std::allocator<T>;

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_BUFFER_ALLOCATOR_HPP
```

11.29 include/CL/sycl/command_group/detail/task.hpp File Reference

```
#include <condition_variable>
#include <memory>
#include <mutex>
#include <thread>
#include <vector>
#include <boost/compute.hpp>
#include "CL/sycl/accessor/detail/accessor_base.hpp"
#include "CL/sycl/buffer/detail/buffer_base.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for task.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::task

  The abstraction to represent SYCL tasks executing inside command_group.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.30 task.hpp

```
00001 #ifndef TRISYCL_SYCL_TASK_HPP
00002 #define TRISYCL_SYCL_TASK_HPP
00003
00004 /** \file The concept of task behind the scene
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <condition_variable>
00013 #include <memory>
00014 #include <mutex>
00015 #include <thread>
00016 #include <vector>
00017
00018 #ifdef TRISYCL_OPENCL
00019 #include <boost/compute.hpp>
00020 #endif
00021
00022 #include "CL/sycl/accessor/detail/accessor_base.hpp"
00023 #include "CL/sycl/buffer/detail/buffer_base.hpp"
00024 #include "CL/sycl/detail/debug.hpp"
00025 #include "CL/sycl/kernel.hpp"
00026 #include "CL/sycl/queue/detail/queue.hpp"
```

```

namespace cl {
namespace sycl {
namespace detail {

/** The abstraction to represent SYCL tasks executing inside command_group

    "enable_shared_from_this" allows to access the shared_ptr behind the
    scene.
*/
struct task : public std::enable_shared_from_this<task>,
              public detail::debug<task> {

  /** List of the buffers used by this task

      \todo Use a set to check that some buffers are not used many
      times at least on writing
  */
  std::vector<std::shared_ptr<detail::buffer_base>> buffers_in_use;

  /// The tasks producing the buffers used by this task
  std::vector<std::shared_ptr<detail::task>> producer_tasks;

  /// Keep track of any prologue to be executed before the kernel
  std::vector<std::function<void(void)>> prologues;

  /// Keep track of any epilogue to be executed after the kernel
  std::vector<std::function<void(void)>> epilogues;

  /// Store if the execution ended, to be notified by task_ready
  bool execution_ended = false;

  /// To signal when this task is ready
  std::condition_variable ready;

  /// To protect the access to the condition variable
  std::mutex ready_mutex;

  /** Keep track of the queue used to submission to notify kernel completion
      or to run OpenCL kernels on */
  std::shared_ptr<detail::queue> owner_queue;

  /// The OpenCL-compatible kernel run by this task, if any
  std::shared_ptr<detail::kernel> kernel;

  /** The accessors indexed by their creation order

      This is used to relate a kernel parameter of a kernel generated
      by the device compiler to its accessor. */
  std::vector<std::weak_ptr<detail::accessor_base>> accessors;


  /// Create a task from a submitting queue
  task(const std::shared_ptr<detail::queue> &q)
    : owner_queue { q } {}


  /// Add a new task to the task graph and schedule for execution
  void schedule(std::function<void(void)> f) {
    /* To keep a copy of the task shared_ptr after the end of the
       command group, capture it by copy in the following lambda. This
       should be easier in C++17 with move semantics on capture
    */
    auto task = shared_from_this();
    auto execution = [=] {
      // Wait for the required tasks to be ready
      task->wait_for_producers();
      task->prelude();
      TRISYCL_DUMP_T("Execute the kernel");
      // Execute the kernel
      f();
      task->postlude();
      // Release the buffers that have been written by this task
      task->release_buffers();
      // Notify the waiting tasks that we are done
      task->notify_consumers();
      // Notify the queue we are done
      owner_queue->kernel_end();
      TRISYCL_DUMP_T("Task thread exit");
    };
    /* Notify the queue that there is a kernel submitted to the
       queue. Do not do it in the task contructor so that we can deal
       with command group without kernel and if we put it inside the
       thread, the queue may have finished before the thread is
       scheduled */
    owner_queue->kernel_start();
    /* \todo it may be implementable with packaged_task that would
```

```
      deal with exceptions in kernels
    */
#ifndef TRISYCL_NO_ASYNC
    /* If in asynchronous execution mode, execute the functor in a new
       thread */
    std::thread thread(execution);
    TRISYCL_DUMP_T("Task thread started");
    /** Detach the thread since it will synchronize by its own means

        \todo This is an issue if there is an exception in the kernel
    */
    thread.detach();
#else
    // Just a synchronous execution otherwise
    execution();
#endif
  }


  /// Wait for the required producer tasks to be ready
  void wait_for_producers() {
    TRISYCL_DUMP_T("Task " << this << " waits for the producer tasks");
    for (auto &t : producer_tasks)
      t->wait();
    // We can let the producers rest in peace
    producer_tasks.clear();
  }


  /// Release the buffers that have  been used by this task
  void release_buffers() {
    TRISYCL_DUMP_T("Task " << this << " releases the written buffers");
    for (auto b: buffers_in_use)
      b->release();
    buffers_in_use.clear();
  }


  /// Notify the waiting tasks that we are done
  void notify_consumers() {
    TRISYCL_DUMP_T("Notify all the task waiting for this task " << this);
   {
     std::unique_lock<std::mutex> ul { ready_mutex };
     execution_ended = true;
   }
    /* \todo Verify that the memory model with the notify does not
       require some fence or atomic */
    ready.notify_all();
  }


  /** Wait for this task to be ready

      This is to be called from another thread
  */
  void wait() {
    TRISYCL_DUMP_T("The task wait for task " << this << " to end");
    std::unique_lock<std::mutex> ul { ready_mutex };
    ready.wait(ul, [&] { return execution_ended; });
  }


  /** Register a buffer to this task

      This is how the dependency graph is incrementally built.
  */
  void add_buffer(std::shared_ptr<detail::buffer_base> &buf,
                  bool is_write_mode) {
    TRISYCL_DUMP_T("Add buffer " << buf << " in task " << this);
    /* Keep track of the use of the buffer to notify its release at
       the end of the execution */
    buffers_in_use.push_back(buf);
    // To be sure the buffer does not disappear before the kernel can run
    buf->use();

    std::shared_ptr<detail::task> latest_producer;
    if (is_write_mode) {
      /* Set this task as the latest producer of the buffer so that
         another kernel may wait on this task */
      latest_producer = buf->set_latest_producer(shared_from_this());
    }
    else
      latest_producer = buf->get_latest_producer();

    /* If the buffer is to be produced by a task, add the task in the
       producer list to wait on it before running the task core

```

```
      If a buffer is accessed first in write mode and then in read mode,
      the task will add itself as a producer and will wait for itself
      when calling \c wait_for_producers, we avoid this by checking that
      \c latest_producer is not \c this
    */
    if (latest_producer && latest_producer != shared_from_this())
      producer_tasks.push_back(latest_producer);
  }


  /// Execute the prologues
  void prelude() {
    TRISYCL_DUMP_T("task::prelude");

    for (const auto &p : prologues)
      p();
    /* Free the functors that may own an accessor owning a buffer
       preventing the command group to complete */
    prologues.clear();
  }


  /// Execute the epilogues
  void postlude() {
    for (const auto &p : epilogues)
      p();
    /* Free the functors that may own an accessor owning a buffer
       preventing the command group to complete */
    epilogues.clear();
  }


  /// Add a function to the prelude to run before kernel execution
  void add_prelude(const std::function<void(void)> &f) {
    TRISYCL_DUMP_T("task::add_prelude");

    prologues.push_back(f);
  }


  /// Add a function to the postlude to run after kernel execution
  void add_postlude(const std::function<void(void)> &f) {
    epilogues.push_back(f);
  }


  /// Get the queue behind the task to run a kernel on
  auto get_queue() {
    return owner_queue;
  }


  /// Set the OpenCL-compatible kernel running this task if any
  void set_kernel(const std::shared_ptr<cl::sycl::detail::kernel> &k) {
    kernel = k;
  }


  /** Get the kernel running if any

      \todo Specify this error in the spec
  */
  cl::sycl::detail::kernel &get_kernel() {
    if (!kernel)
      throw non_cl_error("Cannot use an OpenCL kernel in this context");
    return *kernel;
  }


  /// Set a kernel argument by address
  void set_arg(int arg_index, std::size_t arg_size, const void *scalar_value) {
#ifdef TRISYCL_OPENCL
    // Forward to the OpenCL kernel
    get_kernel().get_boost_compute().set_arg(arg_index,
                                              arg_size,
                                              scalar_value);
#else
    throw non_cl_error("Not compiled with OpenCL support");
#endif
  }


  /// Set a kernel argument by value
  template <typename T>
  void set_arg(int arg_index, const T &scalar_value) {
#ifdef TRISYCL_OPENCL
    // Forward to the OpenCL kernel
```

```
    get_kernel().get_boost_compute().set_arg(arg_index, scalar_value);
#else
    throw non_cl_error("Not compiled with OpenCL support");
#endif
  }


#ifdef TRISYCL_OPENCL
  /// Get the Boost.Compute buffer for an accessor of the task
  auto get_compute_buffer(std::size_t order) {
    return accessors[order].lock()->get_cl_buffer();
  }
#endif


/** Register an accessor and return its registration order

    I would prefer to name this method \c register but this is a C++
    keyword...
*/
std::size_t register_accessor(std::weak_ptr<detail::accessor_base>
a) {
  accessors.push_back(a);
  return accessors.size() - 1;
}

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_TASK_HPP
```

11.31 include/CL/sycl/context/detail/context.hpp File Reference

```
#include "CL/sycl/device.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/detail/default_classes.hpp"
```

Include dependency graph for context.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::context

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.32 context.hpp

```
00001 #ifndef TRISYCL_SYCL_CONTEXT_DETAIL_CONTEXT_HPP
00002 #define TRISYCL_SYCL_CONTEXT_DETAIL_CONTEXT_HPP
00003
00004 /** \file The OpenCL SYCL abstract context
00005
00006     a-doumoulakis at gmail dot com (Anastasios Doumoulakis)
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
```

```
#include "CL/sycl/device.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/detail/default_classes.hpp"

namespace cl {
namespace sycl {
namespace detail {

  /** \addtogroup execution Platforms, contexts, devices and queues
      @{
  */

class context {
public:

#ifdef TRISYCL_OPENCL
  /// Return the underlying \c cl_context of the \c cl::sycl::context
  virtual cl_context get() const = 0;

  /** Return the underlying \c boost::compute::context
      of the \c cl::sycl::context
  */
  virtual boost::compute::context &get_boost_compute() = 0;

  /** Return the underlying \c boost::compute::command_queue associated
      with the context
  */
  virtual boost::compute::command_queue &get_boost_queue() = 0;
#endif

  /// Returns true is the context is a SYCL host context
  virtual bool is_host() const = 0;

  /// Returns the SYCL platform that the context is initialized for
  virtual cl::sycl::platform get_platform() const = 0;

  /** \todo virtual cannot be templated
      template <info::context Param>
      typename info::param_traits<info::context, Param>::type
      get_info() const = 0;
  */

  /// Returns the set of devices that are part of this context.
  virtual vector_class<cl::sycl::device>
get_devices() const = 0;

  /// Virtual to call the real destructor
  virtual ~context() {}

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
```

11.33 include/CL/sycl/context.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/context/detail/host_context.hpp"
#include "CL/sycl/context/detail/opencl_context.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
```

```
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/context.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for context.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::context

  SYCL context. More...
- struct std::hash< cl::sycl::context >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

11.34 context.hpp

```
#ifndef TRISYCL_SYCL_CONTEXT_HPP
#define TRISYCL_SYCL_CONTEXT_HPP

/** \file The OpenCL SYCL context

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/context/detail/host_context.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/context/detail/opencl_context.hpp"
#endif

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/context.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** SYCL context

    The context class encapsulates an OpenCL context, which is implicitly
    created and the lifetime of the context instance defines the lifetime
    of the underlying OpenCL context instance.

    On destruction clReleaseContext is called.

    The default context is the SYCL host context containing only the SYCL
    host device.

    \todo The implementation is quite minimal for now.
*/
class context

  /* Use the underlying context implementation that can be shared in the
     SYCL model */
  : public detail::shared_ptr_implementation<context, detail::context> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<context, detail::context>
;

public:

    // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** Constructs a context object for SYCL host using an async_handler for
      handling asynchronous errors

      Note that the default case asyncHandler = nullptr is handled by the
      default constructor.
  */
  explicit context(async_handler asyncHandler) {
    detail::unimplemented();
  }


#ifdef TRISYCL_OPENCL
  /** Make a SYCL context from an OpenCL context

      The constructor executes a retain on the \c cl_context.

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the \c async_handler, if
      provided.
  */
  context(cl_context clContext, async_handler asyncHandler = nullptr)
```

```
    : context { boost::compute::context { clContext }, asyncHandler } {}


  /// Build a SYCL context from a Boost.Compute context
  context(const boost::compute::context &c,
          async_handler asyncHandler = nullptr)
    : implementation_t { detail::opencl_context::instance(c) } {}
#endif

  /** Constructs a context object using a device_selector object

      The context is constructed with a single device retrieved from the
      device_selector object provided.

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the async_handler, if provided.
  */
  context(const device_selector &deviceSelector,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }


  /** Constructs a context object using a device object

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the async_handler, if provided.
  */
  context(const device &dev,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }


  /** Constructs a context object using a platform object

      Return synchronous errors via the SYCL exception class and
      asynchronous errors are handled via the async_handler, if provided.
  */
  context(const platform &plt,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }


  /* Constructs a context object using a vector_class of device objects

     Return synchronous errors via the SYCL exception class and
     asynchronous errors are handled via the async_handler, if provided.

     \todo Update the specification to replace vector by collection
     concept.
  */
  context(const vector_class<device> &deviceList,
          async_handler asyncHandler = nullptr) {
    detail::unimplemented();
  }

  /** Default constructor that chooses the context according the
      heuristics of the default selector

      Return synchronous errors via the SYCL exception class.

      Get the default constructors back.
  */
  context() : implementation_t { detail::host_context::instance() } {}


#ifdef TRISYCL_OPENCL
  /** Return the underlying \c cl_context object, after retaining
      the \c cl_context.

     Retains a reference to the returned \c cl_context object.

     Caller should release it when finished.
  */
  cl_context get() const {
    return implementation->get();
  }

  /** Return the underlying \c boost::compute::context
      of the \c cl::sycl::context
  */
  boost::compute::context &get_boost_compute() const {
    return implementation->get_boost_compute();
```

```
  }

  /** Return the internal queue that is associated to the context and
      used by triSYCL to move data between some different contests for
      example
  */
  boost::compute::command_queue &get_boost_queue() const {
    return implementation->get_boost_queue();
  }
#endif


  /// Specifies whether the context is in SYCL Host Execution Mode.
  bool is_host() const {
    return implementation->is_host();
  }


  /** Returns the SYCL platform that the context is initialized for

      \todo To be implemented
  */
  platform get_platform();


  /** Returns the set of devices that are part of this context

      \todo To be implemented
  */
  vector_class<device> get_devices() const {
    detail::unimplemented();
    return {};
  }


  /** Queries OpenCL information for the under-lying cl context

      \todo To be implemented
  */
  template <info::context Param>
  typename info::param_traits<info::context, Param>::type
get_info() const {
    detail::unimplemented();
    return {};
  }

};

/// @} to end the execution Doxygen group

}
}

namespace std {

  template <> struct hash<cl::sycl::context> {
    auto operator()(const cl::sycl::context &c) const {
    return c.hash();
    }
  };

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_CONTEXT_HPP
```

11.35 include/CL/sycl/info/context.hpp File Reference

```
#include "CL/sycl/info/param_traits.hpp"
```

Include dependency graph for context.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Enumerations

- enum cl::sycl::info::context : int { cl::sycl::info::context::reference_count, cl::sycl::info::context::platform, cl::sycl::info::context::devices }

  Context information descriptors.

11.36 context.hpp

```
#ifndef TRISYCL_SYCL_INFO_CONTEXT_HPP
#define TRISYCL_SYCL_INFO_CONTEXT_HPP

/** \file The OpenCL SYCL context information parameters

    Anastasi at Xilinx dot com

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/info/param_traits.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/
namespace info {

/** Context information descriptors

    \todo Should be unsigned int to be consistent with others?
*/
enum class context : int {
  reference_count,
  platform,
  devices
};


/** Query the return type for get_info() on context stuff

    \todo To be implemented, return always void.
*/
TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::context, void)

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/


#endif //TRISYCL_SYCL_INFO_CONTEXT_HPP
```

11.37 include/CL/sycl/context/detail/host_context.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
```

```
#include "CL/sycl/context/detail/context.hpp"
```

Include dependency graph for host_context.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::host_context

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.38 host_context.hpp

```
#ifndef TRISYCL_SYCL_CONTEXT_DETAIL_HOST_CONTEXT_HPP
#define TRISYCL_SYCL_CONTEXT_DETAIL_HOST_CONTEXT_HPP

/** \file The OpenCL SYCL host context implementation

    a-doumoulakis at gmail dot com (Anastasios Doumoulakis)

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
#include "CL/sycl/context/detail/context.hpp"

namespace cl {
namespace sycl {
namespace detail {


/** \addtogroup execution Platforms, contexts, devices and queues
    @{

    SYCL host context

    \todo The implementation is quite minimal for now. :-)
*/
class host_context : public detail::context,
                     public detail::singleton<host_context> {

public:

#ifdef TRISYCL_OPENCL
  /** Return the underlying \c cl_context of the \c cl::sycl::context

      This throws an error since there is no OpenCL context associated
      to the host device.
  */
  cl_context get() const override {
    throw non_cl_error("The host context has no OpenCL context");
  }


  /** Return the SYCL platform that the context is initialized for

      This throws an error since there is no \c boost::compute context
      associated to the host device.
  */
  boost::compute::context &get_boost_compute() override {
    throw non_cl_error("The host context has no OpenCL context");
  }


  /** Return the internal OpenCL queue that is associated to the host
      context

      This throws an error since there is no \c
      boost::compute::command_queue associated to the host context.
  */
  boost::compute::command_queue &get_boost_queue() override {
    throw non_cl_error("The host context cannot have an OpenCL queue");
  }
#endif


  /// Return true since the context is a SYCL host context
  bool is_host() const override {
    return true;
  }


  /** Return the platform of the context

      Return synchronous errors via the SYCL exception class.
  */
  cl::sycl::platform get_platform() const override {
    // Return the host platform
    return {};
  }

```

```
#if 0
  /** Query the context for OpenCL \c info::context info

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  template <info::context Param>
  typename info::param_traits<info::context, Param>::type
  get_info() const override {
    detail::unimplemented();
    return {};
  }
#endif


  /** Returns the set of devices that are part of this context.
      It should only return the host device itself.

      \todo To be implemented
  */
  vector_class<cl::sycl::device>
  get_devices() const override {
    // Return just the host device
    return { {} };
  }
};

/// @} to end the execution Doxygen group

}
}
}
#endif // TRISYCL_SYCL_CONTEXT_DETAIL_HOST_CONTEXT_HPP
```

11.39 include/CL/sycl/context/detail/opencl_context.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/context/detail/context.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/exception.hpp"
```

Include dependency graph for opencl_context.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::opencl_context

  SYCL OpenCL context.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Variables

- TRISYCL_WEAK_ATTRIB_PREFIX detail::cache< cl_context, detail::opencl_context > opencl_context↩ ::cache cl::sycl::detail::TRISYCL_WEAK_ATTRIB_SUFFIX

11.40 opencl_context.hpp

```
#ifndef TRISYCL_SYCL_CONTEXT_DETAIL_OPENCL_CONTEXT_HPP
#define TRISYCL_SYCL_CONTEXT_DETAIL_OPENCL_CONTEXT_HPP

/** \file The SYCL OpenCL context implementation

    a-doumoulakis at gmail dot com (Anastasios Doumoulakis)

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/


#include <boost/compute.hpp>

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/detail/cache.hpp"

#include "CL/sycl/context/detail/context.hpp"
#include "CL/sycl/platform.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/exception.hpp"


namespace cl {
namespace sycl {
namespace detail {

/// SYCL OpenCL context
class opencl_context : public detail::context {

  /// User the Boost Compute abstraction of the OpenCL context
  boost::compute::context c;

  /** A boost \c command_queue associated to an OpenCL context for
      when we need to transfer data but no queue is given
      (eg. When an buffer accessor is created)
  */
  boost::compute::command_queue q;

  /** A cache to always return the same alive context for a given OpenCL
      context

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_context, detail::opencl_context>
 cache;

public:

  /// Return the underlying \c cl_context of the \c cl::sycl::context
  cl_context get() const override {
    return c.get();
  }


  /** Return the underlying \c boost::compute::context
      of the \c cl::sycl::context
  */
  boost::compute::context &get_boost_compute() override {
    return c;
  }


  /// Return the queue that is associated to the context
  boost::compute::command_queue &get_boost_queue() override {
    return q;
  }


  /// Return false because the context is not a SYCL host context
  bool is_host() const override {
    return false;
  }

#if 0
  /** Query the context for OpenCL \c info::context info

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  template <info::context Param>
  typename info::param_traits<info::context, Param>::type
```

```
  get_info() const override{
    detail::unimplemented();
    return {};
  }
#endif

  /** Return the platform of the context

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
  cl::sycl::platform get_platform() const override {
    detail::unimplemented();
    return {};
  }


  /** Returns the set of devices that are part of this context.

      \todo To be implemented
  */
  vector_class<cl::sycl::device> get_devices() const override {
    detail::unimplemented();
    return {};
  }

  /// Get a singleton instance of the \c opencl_context
  static std::shared_ptr<opencl_context>
  instance(const boost::compute::context &c) {
    return cache.get_or_register(c.get(),
                                 [&] { return new opencl_context { c }; });
  }


private:

  /// Only the instance factory can build it
  opencl_context(const boost::compute::context &c) :
    c { c },
    q { boost::compute::command_queue { c, c.get_device() } } {}


public:
  /// Unregister from the cache on destruction
  ~opencl_context() override {
    cache.remove(c.get());
  }

};


/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
TRISYCL_WEAK_ATTRIB_PREFIX
detail::cache<cl_context, detail::opencl_context>
opencl_context::cache
TRISYCL_WEAK_ATTRIB_SUFFIX;

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif //TRISYCL_SYCL_CONTEXT_DETAIL_OPENCL_CONTEXT_HPP
```

11.41 include/CL/sycl/detail/array_tuple_helpers.hpp File Reference

Some helpers to do array-tuple conversions.

```
#include <tuple>
#include <utility>
```

Include dependency graph for array_tuple_helpers.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::expand_to_vector< V, Tuple, expansion >

  Allows optional expansion of a 1-element tuple to a V::dimension tuple to replicate scalar values in vector initialization. More...
- struct cl::sycl::detail::expand_to_vector< V, Tuple, true >

  Specialization in the case we ask for expansion. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename V , typename Tuple , size_t... Is>
  std::array< typename V::element_type, V::dimension > cl::sycl::detail::tuple_to_array_iterate (Tuple t, std↩ ::index_sequence< Is... >)

  Helper to construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::tuple_to_array (Tuple t)

  Construct an array from initializer elements provided as a tuple.
- template<typename V , typename Tuple >
  auto cl::sycl::detail::expand (Tuple t)

  Create the array data of V from a tuple of initializer.

11.41.1 Detailed Description

Some helpers to do array-tuple conversions.

Used for example to implement cl::sycl::vec<> class.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file array_tuple_helpers.hpp.

11.42 array_tuple_helpers.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
#define TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP

/** \file

    Some helpers to do array-tuple conversions

    Used for example to implement cl::sycl::vec<> class.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <tuple>
#include <utility>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup array_tuple_helpers Helpers to do array and tuple conversion

    @{
*/

/** Helper to construct an array from initializer elements provided as a
    tuple

    The trick is to get the std::index_sequence<> that represent 0,
    1,..., dimension-1 as a variadic template pack Is that we can
    iterate on, in this function.
*/
template <typename V, typename Tuple, size_t... Is>
std::array<typename V::element_type, V::dimension>
tuple_to_array_iterate(Tuple t, std::index_sequence<Is...>) {
  /* The effect is like a static for-loop with Is counting from 0 to
     dimension-1 and thus constructing a uniform initialization { }
```

```
    construction from each tuple element:
    { std::get<0>(t), std::get<1>(t), ..., std::get<dimension-1>(t) }

    The static cast is here to avoid the warning when there is a loss
    of precision, for example when initializing an int from a float.
  */
  return { { static_cast<typename V::element_type>(std::get<Is>(t))...} };
}


/** Construct an array from initializer elements provided as a tuple
 */
template <typename V, typename Tuple>
auto tuple_to_array(Tuple t) {
  /* Construct an index_sequence with 0, 1, ..., (size of the tuple-1)
     so that tuple_to_array_iterate can statically iterate on it */
  return tuple_to_array_iterate<V>(t,
                                   std::make_index_sequence<std::tuple_size<Tuple>::value>{});
}


/** Allows optional expansion of a 1-element tuple to a V::dimension
    tuple to replicate scalar values in vector initialization
*/
template <typename V, typename Tuple, bool expansion = false>
struct expand_to_vector {
  static_assert(V::dimension == std::tuple_size<Tuple>::value,
                "The number of elements in initialization should match the dimension of the vector");

  // By default, act as a pass-through and do not do any expansion
  static auto expand(Tuple t) { return t; }

};


/** Specialization in the case we ask for expansion */
template <typename V, typename Tuple>
struct expand_to_vector<V, Tuple, true> {
  static_assert(std::tuple_size<Tuple>::value == 1,
                "Since it is a vector initialization from a scalar there should be only one initializer value");


  /** Construct a tuple from a value

      \param value is used to initialize each tuple element

      \param size is the number of elements of the tuple to be generated

      The trick is to get the std::index_sequence<> that represent 0,
      1,..., dimension-1 as a variadic template pack Is that we can
      iterate on, in this function.
  */
  template <typename Value, size_t... Is>
  static auto fill_tuple(Value e, std::index_sequence<Is...>) {
    /* The effect is like a static for-loop with Is counting from 0 to
       dimension-1 and thus replicating the pattern to have
       make_tuple( (0, e), (1, e), ... (n - 1, e) )

       Since the "," operator is just here to throw away the Is value
       (which is needed for the pack expansion...), at the end this is
       equivalent to:
       make_tuple( e, e, ..., e )
    */
    return std::make_tuple(((void)Is, e)...);
  }


  /** We expand the 1-element tuple by replicating into a tuple with the
      size of the vector */
  static auto expand(Tuple t) {
    return fill_tuple(std::get<0>(t),
                      std::make_index_sequence<V::dimension>{});
  }

};


/** Create the array data of V from a tuple of initializer

    If there is only 1 initializer, this is a scalar initialization of a
    vector and the value is expanded to all the vector elements first.
*/
template <typename V, typename Tuple>
auto expand(Tuple t) {
  return tuple_to_array<V>(expand_to_vector<V,
                           decltype(t),
```

```
                            /* Only ask the expansion to all vector
                               element if there only a scalar
                               initializer */
                            std::tuple_size<Tuple>::value == 1>{}.expand(t));
 }

 }
 }
 }

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_ARRAY_TUPLE_HELPERS_HPP
```

11.43 include/CL/sycl/detail/cache.hpp File Reference

```
#include <memory>
#include <mutex>
#include <unordered_map>
```

Include dependency graph for cache.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::cache< Key, Value >

  A simple thread safe cache mechanism to cache std::shared_ptr of values indexed by keys.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.44 cache.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_CACHE_HPP
#define TRISYCL_SYCL_DETAIL_CACHE_HPP

/** \file A simple thread-safe cache

    Ronan at Keryell point FR

```

```
    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>
#include <mutex>
#include <unordered_map>

namespace cl {
namespace sycl {
namespace detail {


/** A simple thread safe cache mechanism to cache std::shared_ptr of
    values indexed by keys

    Since internally only std::weak_ptr are stored, this does not
    prevent object deletion but it is up to the programmer not to use
    this cache to retrieve deleted objects.
*/
template <typename Key, typename Value>
class cache {

public:

  /// The type of the keys used to indexed the cache
  using key_type = Key;

  /// The base type of the values stored in the cache
  using value_type = Value;

private:

  /// The caching storage
  std::unordered_map<key_type, std::weak_ptr<value_type>> c;

  /// To make the cache thread-safe
  std::mutex m;

public:

  /** Get a value stored in the cache if present or insert by calling
      a generator function

      \param[in] k is the key used to retrieve the value

      \param[in] create_element is the function to be called if the
      key is not found in the cache to generate a value which is
      inserted for the key. This function has to produce a value
      convertible to a shared_ptr

      \return a shared_ptr to the value retrieved or inserted
  */
  template <typename Functor>
  std::shared_ptr<value_type> get_or_register(const key_type &k,
                                              Functor &&create_element) {
    std::lock_guard<std::mutex> lg { m };

    auto i = c.find(k);
    if (i != c.end())
      if (auto observe = i->second.lock())
        // Returns \c shared_ptr only if target object is still alive
        return observe;

    // Otherwise create and insert a new element
    std::shared_ptr<value_type> e { create_element() };
    c.insert({ k, e });
    return e;
  }


  /** Remove an entry from the cache

      \param[in] k is the key associated to the value to remove from
      the cache
  */
  void remove(const key_type &k) {
    std::lock_guard<std::mutex> lg { m };
    c.erase(k);
  }

};

}
}
}

```

```
/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_CACHE_HPP
```

11.45 include/CL/sycl/detail/container_element_aspect.hpp File Reference

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::container_element_aspect< T >

  A mix-in to add some container element aspects. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.46 container_element_aspect.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_CONTAINER_ELEMENT_ASPECT_HPP
#define TRISYCL_SYCL_DETAIL_CONTAINER_ELEMENT_ASPECT_HPP

/** \file Implement basic types à la STL related to container
    elements, such as value_type, reference...

    Ronan at Keryell point FR

```

```
    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
    @{
*/

/// A mix-in to add some container element aspects
template <typename T>
struct container_element_aspect {

  using value_type = T;
  using pointer = value_type*;
  using const_pointer = const value_type*;
  using reference = value_type&;
  using const_reference = const value_type&;

};

/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_CONTAINER_ELEMENT_ASPECT_HPP
```

11.47 include/CL/sycl/detail/debug.hpp File Reference

```
#include <iostream>
#include <sstream>
#include <string>
#include <thread>
#include <boost/log/trivial.hpp>
#include <boost/type_index.hpp>
```

Include dependency graph for debug.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::debug< T >

  Class used to trace the construction, copy-construction, move-construction and destruction of classes that inherit from it. More...
- struct cl::sycl::detail::display_vector< T >

  Class used to display a vector-like type of classes that inherit from it. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_INTERNAL_DUMP(expression)

  Dump a debug message in a formatted way.
- #define TRISYCL_DUMP(expression) TRISYCL_INTERNAL_DUMP(expression)
- #define TRISYCL_DUMP_T(expression)

  Same as TRISYCL_DUMP() but with thread id first.

Functions

- template<typename KernelName , typename Functor >
  auto cl::sycl::detail::trace_kernel (Functor f)

  Wrap a kernel functor in some tracing messages to have start/stop information when TRISYCL_TRACE_KERNEL macro is defined.

11.47.1 Macro Definition Documentation

11.47.1.1 TRISYCL_DUMP

```
#define TRISYCL_DUMP(
              expression ) TRISYCL_INTERNAL_DUMP(expression)
```

Definition at line 43 of file debug.hpp.

11.47.1.2 TRISYCL_DUMP_T

```
#define TRISYCL_DUMP_T(
              expression )
```

Value:

```
TRISYCL_DUMP("Thread " << std::hex                                  \
              << std::this_thread::get_id() << ": " << expression)
```

Same as TRISYCL_DUMP() but with thread id first.

Definition at line 46 of file debug.hpp.

Referenced by cl::sycl::drt::accessor< Accessor >::accessor(), cl::sycl::detail::accessor< T, Dimensions, Mode, Target >::accessor(), cl::sycl::detail::task::add_buffer(), cl::sycl::detail::task::add_prelude(), cl::sycl::detail::pipe↩_reservation< PipeAccessor >::commit(), cl::sycl::detail::pipe< value_type >::empty(), cl::sycl::detail::queue↩::kernel_end(), cl::sycl::detail::queue::kernel_start(), cl::sycl::detail::task::notify_consumers(), cl::sycl::detail::pipe↩_reservation< PipeAccessor >::operator[ ](), cl::sycl::detail::task::prelude(), cl::sycl::drt::code::program::program(), cl::sycl::detail::pipe< value_type >::read(), cl::sycl::detail::task::release_buffers(), cl::sycl::detail::pipe< value_↩type >::reserve_read(), cl::sycl::detail::pipe< value_type >::reserve_write(), cl::sycl::detail::task::schedule(), cl↩::sycl::handler::schedule_kernel(), cl::sycl::drt::serialize_accessor_arg(), cl::sycl::drt::serialize_arg(), cl::sycl::drt↩::set_kernel(), cl::sycl::handler::single_task(), cl::sycl::detail::pipe< value_type >::size(), cl::sycl::detail::task::wait(), cl::sycl::detail::queue::wait_for_kernel_execution(), cl::sycl::detail::task::wait_for_producers(), cl::sycl::detail::pipe< value_type >::write(), and cl::sycl::detail::buffer_waiter< T, Dimensions, Allocator >::~buffer_waiter().

11.47.1.3 TRISYCL_INTERNAL_DUMP

```
#define TRISYCL_INTERNAL_DUMP(
              expression )
```

Value:

```
do {        \
    std::ostringstream s;                           \
    s << expression;                                \
    BOOST_LOG_TRIVIAL(debug) << s.str();            \
  } while(0)
```

Dump a debug message in a formatted way.

Use an intermediate ostringstream because there are issues with BOOST_LOG_TRIVIAL to display C strings

Definition at line 35 of file debug.hpp.

Referenced by cl::sycl::detail::trace_kernel().

11.48 debug.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_DEBUG_HPP
#define TRISYCL_SYCL_DETAIL_DEBUG_HPP

/** \file Track constructor/destructor invocations and trace kernel execution

    Define the TRISYCL_DEBUG CPP flag to have an output.

    To use it in some class C, make C inherit from debug<C>.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <iostream>

// The common debug and trace infrastructure
#if defined(TRISYCL_DEBUG) || defined(TRISYCL_TRACE_KERNEL)
#include <sstream>
#include <string>
#include <thread>

#include <boost/log/trivial.hpp>
#include <boost/type_index.hpp>

// To be able to construct string literals like "blah"s
using namespace std::string_literals;

/** Dump a debug message in a formatted way.

    Use an intermediate ostringstream because there are issues with
    BOOST_LOG_TRIVIAL to display C strings
*/
#define TRISYCL_INTERNAL_DUMP(expression) do {        \
    std::ostringstream s;                             \
    s << expression;                                  \
    BOOST_LOG_TRIVIAL(debug) << s.str();              \
  } while(0)
#endif

#ifdef TRISYCL_DEBUG
#define TRISYCL_DUMP(expression) TRISYCL_INTERNAL_DUMP(expression)

/// Same as TRISYCL_DUMP() but with thread id first
#define TRISYCL_DUMP_T(expression)                                        \
  TRISYCL_DUMP("Thread " << std::hex                                      \
               << std::this_thread::get_id() << ": " << expression)
#else
#define TRISYCL_DUMP(expression) do { } while(0)
#define TRISYCL_DUMP_T(expression) do { } while(0)
#endif

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup debug_trace Debugging and tracing support
    @{
*/

/** Class used to trace the construction, copy-construction,
    move-construction and destruction of classes that inherit from it

    \param T is the real type name to be used in the debug output.
*/
template <typename T>
struct debug {
  // To trace the execution of the conSTRUCTORs and deSTRUCTORs
#ifdef TRISYCL_DEBUG_STRUCTORS
  /// Trace the construction with the compiler-dependent mangled named
  debug() {
    TRISYCL_DUMP("Constructor of "
                 << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }


  /** Trace the copy construction with the compiler-dependent mangled
      named

      Only add this constructor if T has itself the same constructor,
      otherwise it may prevent the synthesis of default copy
      constructor and assignment.
```

```
  */
  template <typename U = T>
  debug(debug const &,
        /* Use intermediate U type to have the type dependent for
           enable_if to work

        \todo Use is_copy_constructible_v when moving to C++17 */
        std::enable_if_t<std::is_copy_constructible<U>::value> * = 0) {
    TRISYCL_DUMP("Copy of " << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }


  /** Trace the move construction with the compiler-dependent mangled
      named

      Only add this constructor if T has itself the same constructor,
      otherwise it may prevent the synthesis of default move
      constructor and move assignment.
  */
  template <typename U = T>
  debug(debug &&,
        /* Use intermediate U type to have the type dependent for
           enable_if to work

        \todo Use is_move_constructible_v when moving to C++17 */
        std::enable_if_t<std::is_move_constructible<U>::value> * = 0) {
    TRISYCL_DUMP("Move of " << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }


  /// Trace the destruction with the compiler-dependent mangled named
  ~debug() {
    TRISYCL_DUMP("~ Destructor of "
                 << boost::typeindex::type_id<T>().pretty_name()
                 << " " << (void*) this);
  }
#endif
};


/** Wrap a kernel functor in some tracing messages to have start/stop
    information when TRISYCL_TRACE_KERNEL macro is defined */
template <typename KernelName, typename Functor>
auto trace_kernel(Functor f) {
#ifdef TRISYCL_TRACE_KERNEL
  // Inject tracing message around the kernel
  return [=] () mutable {
    /* Since the class KernelName may just be declared and not really
       defined, just use it through a class pointer to have
       typeid().name() not complaining */
    TRISYCL_INTERNAL_DUMP(
      "Kernel started "
      << boost::typeindex::type_id<KernelName *>().pretty_name());
    f();
    TRISYCL_INTERNAL_DUMP(
      "Kernel stopped "
      << boost::typeindex::type_id<KernelName *>().pretty_name());
  };
#else
  // Identity by default
  return f;
#endif
}


/** Class used to display a vector-like type of classes that inherit from
    it

    \param T is the real type name to be used in the debug output.

    Calling the display() method dump the values on std::cout
*/
template <typename T>
struct display_vector {

  /// To debug and test
  void display() const {
#ifdef TRISYCL_DEBUG
    std::cout << boost::typeindex::type_id<T>().pretty_name() << ":";
#endif
    // Get a pointer to the real object
    for (auto e : *static_cast<const T *>(this))
      std::cout << " " << e;
    std::cout << std::endl;
  }
```

```

};

/// @} End the debug_trace Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_DEBUG_HPP
```

11.49 include/CL/sycl/detail/default_classes.hpp File Reference

```
#include <memory>
#include <vector>
#include <string>
#include <functional>
#include <mutex>
#include <exception>
```

Include dependency graph for default_classes.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- template<class T , class Alloc = std::allocator<T>>
  using cl::sycl::vector_class = std::vector< T, Alloc >
- using cl::sycl::string_class = std::string
- template<class R , class... ArgTypes>
  using cl::sycl::function_class = std::function< R(ArgTypes...)>
- using cl::sycl::mutex_class = std::mutex
- template<class T , class D = std::default_delete<T>>
  using cl::sycl::unique_ptr_class = std::unique_ptr< T[ ], D >
- template<class T >
  using cl::sycl::shared_ptr_class = std::shared_ptr< T >
- template<class T >
  using cl::sycl::weak_ptr_class = std::weak_ptr< T >
- template<class T >
  using cl::sycl::hash_class = std::hash< T >
- using cl::sycl::exception_ptr_class = std::exception_ptr

11.50 default_classes.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
#define TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP

/** \file The OpenCL SYCL default classes to use from the STL according to
    section 3.2 of SYCL 1.2 specification

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \addtogroup defaults Manage default configuration and types
    @{
*/

#ifndef CL_SYCL_NO_STD_VECTOR
/** The vector type to be used as SYCL vector
 */
#include <memory>
#include <vector>
namespace cl {
namespace sycl {

template <class T, class Alloc = std::allocator<T>>
using vector_class = std::vector<T, Alloc>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_STRING
/** The string type to be used as SYCL string
 */
#include <string>
namespace cl {
namespace sycl {

using string_class = std::string;

}
}
#endif


#ifndef CL_SYCL_NO_STD_FUNCTION
/** The functional type to be used as SYCL function
 */
#include <functional>
namespace cl {
namespace sycl {

template <class R, class... ArgTypes>
using function_class = std::function<R(ArgTypes...)>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_MUTEX
/** The mutex type to be used as SYCL mutex
 */
#include <mutex>
namespace cl {
namespace sycl {

using mutex_class = std::mutex;

}
}
#endif


#ifndef CL_SYCL_NO_STD_UNIQUE_PTR
/** The unique pointer type to be used as SYCL unique pointer
 */
#include <memory>
namespace cl {
namespace sycl {

template <class T, class D = std::default_delete<T>>
using unique_ptr_class = std::unique_ptr<T[], D>;
```

```

}
}
#endif


#ifndef CL_SYCL_NO_STD_SHARED_PTR
/** The shared pointer type to be used as SYCL shared pointer
 */
#include <memory>
namespace cl {
namespace sycl {

template <class T>
using shared_ptr_class = std::shared_ptr<T>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_WEAK_PTR
/** The weak pointer type to be used as SYCL weak pointer
 */
#include <memory>
namespace cl {
namespace sycl {

template <class T>
using weak_ptr_class = std::weak_ptr<T>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_HASH
/** The hash type to be used as SYCL hash
 */
#include <functional>
namespace cl {
namespace sycl {

template <class T>
using hash_class = std::hash<T>;

}
}
#endif


#ifndef CL_SYCL_NO_STD_EXCEPT
/** The exception pointer type to be used as SYCL exception pointer
 */
#include <exception>
namespace cl {
namespace sycl {

using exception_ptr_class = std::exception_ptr;

}
}
#endif


/// @} End the defaults Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_DEFAULT_CLASSES_HPP
```

11.51 include/CL/sycl/detail/global_config.hpp File Reference

```
#include <CL/cl.h>
#include <boost/compute.hpp>
```

Include dependency graph for global_config.hpp:

This graph shows which files directly or indirectly include this file:

Macros

- #define CL_SYCL_LANGUAGE_VERSION 121

  This implements SYCL 1.2.1.
- #define TRISYCL_CL_LANGUAGE_VERSION 121

  This implements triSYCL 1.2.1.
- #define __SYCL_SINGLE_SOURCE__

  This source is compiled by a single source compiler.
- #define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
- #define SDX_KERNEL_PROGRAM_OWNING_BUG

  Define TRISYCL_OPENCL to add OpenCL.
- #define TRISYCL_SKIP_OPENCL(x) x

  A macro to keep some stuff in OpenCL mode.
- #define TRISYCL_DEVICE_ONLY(X)
- #define TRISYCL_WEAK_ATTRIB_PREFIX
- #define TRISYCL_WEAK_ATTRIB_SUFFIX __attribute__((weak))

11.51.1 Macro Definition Documentation

11.51.1.1 TRISYCL_WEAK_ATTRIB_PREFIX

```
#define TRISYCL_WEAK_ATTRIB_PREFIX
```

Definition at line 79 of file global_config.hpp.

Referenced by cl::sycl::detail::__attribute__(), cl::sycl::drt::accessor< Accessor >::operator[](), cl::sycl::drt::code↩::program::program(), cl::sycl::drt::serialize_arg(), cl::sycl::detail::opencl_kernel::TRISYCL_ParallelForKernel_R↩ANGE(), cl::sycl::detail::opencl_context::~opencl_context(), cl::sycl::detail::opencl_device::~opencl_device(), cl↩::sycl::detail::opencl_platform::~opencl_platform(), and cl::sycl::detail::opencl_queue::~opencl_queue().

11.51.1.2 TRISYCL_WEAK_ATTRIB_SUFFIX

```
#define TRISYCL_WEAK_ATTRIB_SUFFIX __attribute__((weak))
```

Definition at line 80 of file global_config.hpp.

Referenced by cl::sycl::device::get_platform(), cl::sycl::drt::accessor< Accessor >::operator[](), cl::sycl::drt::code↩::program::program(), and cl::sycl::drt::serialize_arg().

11.52 global_config.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
#define TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP

/** \file The OpenCL SYCL details on the global triSYCL configuration

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \addtogroup defaults Manage default configuration and types
    @{
*/

// The following symbols can be set to implement a different version
#ifndef CL_SYCL_LANGUAGE_VERSION
/// This implements SYCL 1.2.1
#define CL_SYCL_LANGUAGE_VERSION 121
#endif

#ifndef TRISYCL_CL_LANGUAGE_VERSION
/// This implements triSYCL 1.2.1
#define TRISYCL_CL_LANGUAGE_VERSION 121
#endif

/// This source is compiled by a single source compiler
#define __SYCL_SINGLE_SOURCE__


/* Work-around an old Boost.CircularBuffer bug if a pre 1.62 Boost
   version is used */
#define TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE


/** Define TRISYCL_OPENCL to add OpenCL

    triSYCL can indeed work without OpenCL if only host support is needed.
*/
#ifdef TRISYCL_OPENCL

/* There is a bug in Xilinx OpenCL runtime where the cl_kernel does
   not keep alive its own cl_program */
#define SDX_KERNEL_PROGRAM_OWNING_BUG

// SYCL interoperation API with OpenCL requires some OpenCL C types:
#if defined(__APPLE__)
#include <OpenCL/cl.h>
#else
#include <CL/cl.h>
#endif
// But the triSYCL OpenCL implementation is actually based on Boost.Compute
#include <boost/compute.hpp>
/// A macro to keep some stuff in OpenCL mode
#define TRISYCL_SKIP_OPENCL(x) x
#else
/// A macro to skip stuff when not supporting OpenCL
#define TRISYCL_SKIP_OPENCL(x)
#endif

#ifdef TRISYCL_DEVICE
/// Define the SYCL marker according to specification
#define __SYCL_DEVICE_ONLY__
// To keep something when in device mode
#define TRISYCL_DEVICE_ONLY(X) X
#else
// Do not keep when not in device mode
#define TRISYCL_DEVICE_ONLY(X)
#endif

/// @} End the defaults Doxygen group

// Compiler specific weak linking (until changing to C++17 inline variables/functions)
#ifndef TRISYCL_WEAK_ATTRIB_PREFIX
#ifdef _MSC_VER
#define TRISYCL_WEAK_ATTRIB_PREFIX __declspec(selectany)
#define TRISYCL_WEAK_ATTRIB_SUFFIX
#else
#define TRISYCL_WEAK_ATTRIB_PREFIX
#define TRISYCL_WEAK_ATTRIB_SUFFIX __attribute__((weak))
#endif
#endif

// Suppress usage/leak of macros originating from Visual C++ headers
```

```
#ifdef _MSC_VER
#define NOMINMAX
#endif

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_GLOBAL_CONFIG_HPP
```

11.53 include/CL/sycl/detail/instantiate_kernel.hpp File Reference

```
#include <tuple>
#include "CL/sycl/command_group/detail/task.hpp"
```

Include dependency graph for instantiate_kernel.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename KernelName , typename Functor >
  cl::sycl::detail::__attribute__ ((noinline)) void instantiate_kernel(Functor f) noexcept

  Instantiate the kernel code.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::detail::set_kernel_task_marker (detail::task &) noexcept

  Set the task to be used to launch the kernel.

Variables

- auto cl::sycl::detail::prevent_arguments_from_optimization

  Avoid the interprocedural optimization to remove these arguments in the kernel instantiation by relying on some dummy static variables.

11.54 instantiate_kernel.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_INSTANTIATE_KERNEL_HPP
#define TRISYCL_SYCL_DETAIL_INSTANTIATE_KERNEL_HPP

/** \file Some helpers to share information between the SYCL program
    and the device compiler.

    For example, this brings a function to outline a SYCL kernel at a
    specific place.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <tuple>

#include "CL/sycl/command_group/detail/task.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** Avoid the interprocedural optimization to remove these arguments
    in the kernel instantiation by relying on some dummy static variables.

    Using \c __attribute__((used)) does not work on arguments but only
    on static variable, so use this function.
*/
auto prevent_arguments_from_optimization = [] (auto & ...args) {
  /* Just keep track of the address of all the given objects,
     otherwise the objects are copied, may throw, are registered for
     destruction with \c atexit(), etc. */
  static auto __attribute__((used)) keep = std::make_tuple(& args...);
};


/** Instantiate the kernel code

    To have a clear view of what a kernel is at the LLVM IR level even
    when a lot of optimizations are done (-O3), call the kernel from
    this function marked as \c noinline, otherwise the kernel code is
```

```
    inlined directly where it is used.

    Actually some optimizations/code restructuring might duplicate the
    call to \c instantiate_kernel but the \c noduplicate attribute
    seems incompatible with \c noinline in Clang/LLVM 3.9.1, because
    the inlining happen when both attributes are used...
*/
template <typename KernelName,
          typename Functor>
__attribute__((noinline)) void
instantiate_kernel(Functor f) noexcept {
  /* The outlining compiler is expected to do some massage here or
     around and to insert some calls to \c serialize_arg and so on */
  f();
}


/** Set the task to be used to launch the kernel

    This is actually a dummy function so that the device compiler can
    just grab the task for passing it to the argument serialization
    functions from the device runtime.

    It is better to use a weak linkage than having this function with
    extern linkage:

    - if the device compiler is used the function is kept by the
      compiler because full link and LTO is not applied before the
      device compiler passes are applied. So there is no partial
      evaluation in the compiler according to what is returned by this
      function;

    - if the device compiler is not used, the code still links and can
      be compiled.
*/
TRISYCL_WEAK_ATTRIB_PREFIX void
TRISYCL_WEAK_ATTRIB_SUFFIX
set_kernel_task_marker(detail::task &) noexcept {
}


/** Gather in a single function what is necessary for the device
    compiler to outline the kernel or to set up the kernel and
    serialize the arguments.

    The \c noinline attribute is used to force this to appear as a
    real function the device compiler can massage.
*/
template <typename KernelName,
          typename Kernel>
__attribute__((noinline))
void launch_device_kernel(detail::task &t,
                          Kernel k) noexcept {
  /** Setup the task to be used to launch the kernel.

      This is used by the outlining compiler. */
  detail::set_kernel_task_marker(t);
  /* A simplified version for the device just to be able to extract
     the kernel itself.

     It is not a real outlining but a good approximation to be
     massaged by a compiler later.

     Put the iteration space loops in the kernels for now, which makes
     sense for CPU emulation or FPGA pipelined execution.
  */
  instantiate_kernel<KernelName>(k);
}

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_INSTANTIATE_KERNEL_HPP
```

11.55 include/CL/sycl/detail/linear_id.hpp File Reference

```
#include <cstddef>
```

Include dependency graph for linear_id.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<typename Range , typename Id >
  size_t constexpr cl::sycl::detail::linear_id (Range range, Id id, Id offset={})

  Compute a linearized array access used in the OpenCL 2 world.

11.56 linear_id.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
#define TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP

/** \file Compute linearized array access

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
    @{
*/

/** Compute a linearized array access used in the OpenCL 2 world

    Typically for the get_global_linear_id() and get_local_linear_id()
    functions.
*/
template <typename Range, typename Id>
size_t constexpr inline linear_id(Range range, Id id, Id offset = {}) {
  auto dims = std::distance(std::begin(range), std::end(range));

  size_t linear_id = 0;
  /* A good compiler should unroll this and do partial evaluation to
     remove the first multiplication by 0 of this Horner evaluation and
     remove the 0 offset evaluation */
    for (int i = dims - 1; i >= 0; --i)
      linear_id = linear_id*range[i] + id[i] - offset[i];

    return linear_id;
  }


/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_LINEAR_ID_HPP
```

11.57 include/CL/sycl/detail/shared_ptr_implementation.hpp File Reference

```
#include <functional>
#include <memory>
#include <boost/operators.hpp>
```

Include dependency graph for shared_ptr_implementation.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >

  Provide an implementation as shared_ptr with total ordering and hashing to be used with algorithms and in (un)ordered containers.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.58 shared_ptr_implementation.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
#define TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP

/** \file Mix-in to add an implementation as shared_ptr with total
    ordering and hashing so that the class can be used with algorithms
    and in (un)ordered containers

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <functional>
#include <memory>

#include <boost/operators.hpp>

namespace cl {
namespace sycl {
namespace detail {

/** Provide an implementation as shared_ptr with total ordering and
    hashing to be used with algorithms and in (un)ordered containers

    To be used, a Parent class wanting an Implementation needs to
    inherit from.

    The implementation ends up in a member really named
    "implementation".

    \code
    public detail::shared_ptr_implementation<Parent, Implementation>
    \endcode

    and also inject in std namespace a specialization for
    \code hash<Parent> \endcode
*/
template <typename Parent, typename Implementation>
struct shared_ptr_implementation : public boost::totally_ordered<Parent> {

  /// The implementation forward everything to this... implementation
  std::shared_ptr<Implementation> implementation;

  /// The implementation directly as a shared pointer
  shared_ptr_implementation(std::shared_ptr<Implementation>
i)
    : implementation { i } {}


  /// The implementation takes the ownership from a raw pointer
  shared_ptr_implementation(Implementation *i) : implementation { i } {}


  /// Keep all other constructors to have usual shared_ptr behaviour
  shared_ptr_implementation() = default;


  /** Equality operator

      This is generalized by boost::equality_comparable from
      boost::totally_ordered to implement the equality comparable
      concept
  */
  bool operator ==(const Parent &other) const {
    return implementation == other.implementation;
  }


  /** Inferior operator

      This is generalized by boost::less_than_comparable from
      boost::totally_ordered to implement the equality comparable
      concept

      \todo Add this to the spec
  */
  bool operator <(const Parent &other) const {
    return implementation < other.implementation;
  }


  /// Forward the hashing for unordered containers to the implementation
  auto hash() const {
```

```
    return std::hash<decltype(implementation)>{}(implementation);
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_SHARED_PTR_IMPLEMENTATION_HPP
```

11.59 include/CL/sycl/detail/singleton.hpp File Reference

```
#include <functional>
#include <memory>
#include <boost/core/null_deleter.hpp>
#include <boost/operators.hpp>
```

Include dependency graph for singleton.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::singleton< T >

  Provide a singleton factory.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.60 singleton.hpp

```
00001 #ifndef TRISYCL_SYCL_DETAIL_SINGLETON_HPP
00002 #define TRISYCL_SYCL_DETAIL_SINGLETON_HPP
00003
00004 /** \file Mix-in to add a singleton implementation with an instance() method
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
```

```
    License. See LICENSE.TXT for details.
*/

#include <functional>
#include <memory>

#include <boost/core/null_deleter.hpp>
#include <boost/operators.hpp>


namespace cl {
namespace sycl {
namespace detail {

/// Provide a singleton factory
template <typename T>
struct singleton {

  /// Get a singleton instance of T
  static std::shared_ptr<T> instance() {
    // C++11 guaranties the static construction is thread-safe
    static T single;
    /** Use a null_deleter since the singleton should not be deleted,
        as allocated in the static area */
    static std::shared_ptr<T> sps { &single,
                                    boost::null_deleter {} };

    return sps;
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_SINGLETON_HPP
```

11.61 include/CL/sycl/detail/small_array.hpp File Reference

```
#include <algorithm>
#include <array>
#include <cstddef>
#include <type_traits>
#include <boost/operators.hpp>
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for small_array.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::small_array< BasicType, FinalType, Dims, EnableArgsConstructor >

  Define a multi-dimensional index, used for example to locate a work item or a buffer element. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, Dims >

  A small array of 1, 2 or 3 elements with the implicit constructors. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >

  Use some specializations so that some function overloads can be determined according to some implicit constructors and to have an implicit conversion from/to BasicType (such as an int typically) if Dimensions = 1. More...
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >
- struct cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)

  Helper macro to declare a vector operation with the given side-effect operator.
- #define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(op)

11.62 small_array.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
#define TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP

/** \file This is a small array class to build range<>, id<>, etc.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <array>
#include <cstddef>
#include <type_traits>

#include <boost/operators.hpp>

#include "CL/sycl/detail/debug.hpp"


namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
    @{
*/


/** Helper macro to declare a vector operation with the given side-effect
    operator */
#define TRISYCL_BOOST_OPERATOR_VECTOR_OP(op)            \
  FinalType operator op(const FinalType &rhs) {         \
    for (std::size_t i = 0; i != Dims; ++i)             \
      (*this)[i] op rhs[i];                             \
    return *this;                                       \
  }


#define TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(op)          \
  FinalType operator op(const FinalType &rhs) {         \
    FinalType res;                                      \
    for (std::size_t i = 0; i != Dims; ++i)             \
      res[i] = (*this)[i] op rhs[i];                    \
    return res;                                         \
  }


/** Define a multi-dimensional index, used for example to locate a work
    item or a buffer element

    Unfortunately, even if std::array is an aggregate class allowing
    native list initialization, it is no longer an aggregate if we derive
    from an aggregate. Thus we have to redeclare the constructors.

    \param BasicType is the type element, such as int

    \param Dims is the dimension number, typically between 1 and 3

    \param FinalType is the final type, such as range<> or id<>, so that
    boost::operator can return the right type

    \param EnableArgsConstructor adds a constructors from Dims variadic
    elements when true. It is false by default.

    std::array<> provides the collection concept, with .size(), == and !=
    too.
*/
template <typename BasicType,
          typename FinalType,
          std::size_t Dims,
          bool EnableArgsConstructor = false>
struct small_array : std::array<BasicType, Dims>,
  // To have all the usual arithmetic operations on this type
  boost::euclidean_ring_operators<FinalType>,
  // Bitwise operations
  boost::bitwise<FinalType>,
  // Shift operations
  boost::shiftable<FinalType>,
  // Already provided by array<> lexicographically:
  // boost::equality_comparable<FinalType>,
  // boost::less_than_comparable<FinalType>,
  // Add a display() method
```

```
detail::display_vector<FinalType> {

/// \todo add this Boost::multi_array or STL concept to the
/// specification?
static const auto dimensionality = Dims;

/* Note that constexpr size() from the underlying std::array provides
   the same functionality */
static const size_t dimension = Dims;
using element_type = BasicType;


/** A constructor from another array

    Make it explicit to avoid spurious range<> constructions from int *
    for example
*/
template <typename SourceType>
small_array(const SourceType src[Dims]) {
  // (*this)[0] is the first element of the underlying array
  std::copy_n(src, Dims, &(*this)[0]);
}


/** An accessor to the first variable of a small array
*/
BasicType& x(){
  static_assert(Dims >= 1, "can't access to small_array[0] if Dims < 1");
  return (*this)[0];
}


/** An accessor to the second variable of a small array
*/
BasicType& y(){
  static_assert(Dims >= 2, "can't access to small_array[1] if Dims < 2");
  return (*this)[1];
}


/** An accessor to the third variable of a small array
*/
BasicType& z(){
  static_assert(Dims >= 3, "can't access to small_array[2] if Dims < 3");
  return (*this)[2];
}


/// A constructor from another small_array of the same size
template <typename SourceBasicType,
          typename SourceFinalType,
          bool SourceEnableArgsConstructor>
small_array(const small_array<SourceBasicType,
            SourceFinalType,
            Dims,
            SourceEnableArgsConstructor> &src) {
  std::copy_n(&src[0], Dims, &(*this)[0]);
}


/** Initialize the array from a list of elements

    Strangely, even when using the array constructors, the
    initialization of the aggregate is not available. So recreate an
    equivalent here.

    Since there are inherited types that defines some constructors with
    some conflicts, make it optional here, according to
    EnableArgsConstructor template parameter.
 */
template <typename... Types,
          // Just to make enable_if depend of the template and work
          bool Depend = true,
          typename = typename std::enable_if_t<EnableArgsConstructor
                                               && Depend>>
small_array(const Types &... args)
  : std::array<BasicType, Dims> {
  // Allow a loss of precision in initialization with the static_cast
  { static_cast<BasicType>(args)... }
}
{
  static_assert(sizeof...(args) == Dims,
                "The number of initializing elements should match "
                "the dimension");
}


```

```
  /// Construct a small_array from a std::array
  template <typename SourceBasicType>
  small_array(const std::array<SourceBasicType, Dims> &src)
  : std::array<BasicType, Dims>(src) {}


  /// Keep other constructors from the underlying std::array
  using std::array<BasicType, Dims>::array;

  /// Keep the synthesized constructors
  small_array() = default;

  /// Return the element of the array
  auto get(std::size_t index) const {
    return (*this)[index];
  }

  /* Implement minimal methods boost::euclidean_ring_operators needs to
     generate everything */
  /// Add + like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(+=)

  /// Add - like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(-=)

  /// Add * like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(*=)

  /// Add / like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(/=)

  /// Add % like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(%=)

  /// Add << like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(<<=)

  /// Add >> like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(>>=)

  /// Add & like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(&=)

  /// Add ^ like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(^=)

  /// Add | like operations on the id<> and others
  TRISYCL_BOOST_OPERATOR_VECTOR_OP(|=)

  TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(&&)

  TRISYCL_LOGICAL_OPERATOR_VECTOR_OP(||)


  /** Since the boost::operator work on the small_array, add an implicit
      conversion to produce the expected type */
  operator FinalType () {
    return *static_cast<FinalType *>(this);
  }

};


/** A small array of 1, 2 or 3 elements with the implicit constructors */
template <typename BasicType, typename FinalType, std::size_t Dims>
struct small_array_123 : small_array<BasicType, FinalType, Dims> {
  static_assert(1 <= Dims && Dims <= 3,
                "Dimensions are between 1 and 3");
};


/** Use some specializations so that some function overloads can be
    determined according to some implicit constructors and to have an
    implicit conversion from/to BasicType (such as an int typically) if
    Dimensions = 1
*/
template <typename BasicType, typename FinalType>
struct small_array_123<BasicType, FinalType, 1>
  : public small_array<BasicType, FinalType, 1> {
  /// A 1-D constructor to have implicit conversion from from 1 integer
  /// and automatic inference of the dimensionality
  small_array_123(BasicType x) {
    (*this)[0] = x;
  }


  /// Keep other constructors
```

```
  small_array_123() = default;

  using small_array<BasicType, FinalType, 1>::small_array;

  /** Conversion so that an for example an id<1> can basically be used
      like an integer */
  operator BasicType() const {
    return (*this)[0];
  }
};


template <typename BasicType, typename FinalType>
struct small_array_123<BasicType, FinalType, 2>
  : public small_array<BasicType, FinalType, 2> {
  /// A 2-D constructor to have implicit conversion from from 2 integers
  /// and automatic inference of the dimensionality
  small_array_123(BasicType x, BasicType y) {
    (*this)[0] = x;
    (*this)[1] = y;
  }


  /** Broadcasting constructor initializing all the elements with the
      same value

      \todo Add to the specification of the range, id...
  */
  explicit small_array_123(BasicType e) : small_array_123 { e, e } { }


  /// Keep other constructors
  small_array_123() = default;

  using small_array<BasicType, FinalType, 2>::small_array;
};


template <typename BasicType, typename FinalType>
struct small_array_123<BasicType, FinalType, 3>
  : public small_array<BasicType, FinalType, 3> {
  /// A 3-D constructor to have implicit conversion from from 3 integers
  /// and automatic inference of the dimensionality
  small_array_123(BasicType x, BasicType y, BasicType z) {
    (*this)[0] = x;
    (*this)[1] = y;
    (*this)[2] = z;
  }


  /** Broadcasting constructor initializing all the elements with the
      same value

      \todo Add to the specification of the range, id...
  */
  explicit small_array_123(BasicType e) : small_array_123 { e, e, e } { }


  /// Keep other constructors
  small_array_123() = default;

  using small_array<BasicType, FinalType, 3>::small_array;
};

/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_SMALL_ARRAY_HPP
```

11.63 include/CL/sycl/detail/SPIR/opencl_spir.h File Reference

Macros

- #define const_func __attribute__((const))

  Added according to pocl/examples/example1-spir64/generate_spir.sh.
- #define readonly __attribute__((pure))
- #define MAXFLOAT 0x1.fffffep127f

  Value of maximum non-infinite single-precision floating-point number.
- #define HUGE_VALF (__builtin_huge_valf())

  A positive float constant expression.
- #define HUGE_VAL (__builtin_huge_val())

  A positive double constant expression.
- #define INFINITY (__builtin_inff())

  A constant expression of type float representing positive or unsigned infinity.
- #define NAN as_float(__builtin_nanf((char const *)""))

  A constant expression of type float representing a quiet NaN.
- #define FP_ILOGB0 INT_MIN
- #define FP_ILOGBNAN INT_MAX
- #define FLT_DIG 6
- #define FLT_MANT_DIG 24
- #define FLT_MAX_10_EXP +38
- #define FLT_MAX_EXP +128
- #define FLT_MIN_10_EXP -37
- #define FLT_MIN_EXP -125
- #define FLT_RADIX 2
- #define FLT_MAX 0x1.fffffep127f
- #define FLT_MIN 0x1.0p-126f
- #define FLT_EPSILON 0x1.0p-23f
- #define DBL_DIG 15
- #define DBL_MANT_DIG 53
- #define DBL_MAX_10_EXP +308
- #define DBL_MAX_EXP +1024
- #define DBL_MIN_10_EXP -307
- #define DBL_MIN_EXP -1021
- #define DBL_RADIX 2
- #define DBL_MAX 0x1.fffffffffffffp1023
- #define DBL_MIN 0x1.0p-1022
- #define DBL_EPSILON 0x1.0p-52
- #define M_E_F 2.71828182845904523536028747135266250f
- #define M_LOG2E_F 1.44269504088896340735992468100189214f
- #define M_LOG10E_F 0.43429448190325182765112891891660508 2f
- #define M_LN2_F 0.69314718055994530941723212145817656 8f
- #define M_LN10_F 2.30258509299404568401799145468436421f
- #define M_PI_F 3.14159265358979323846264338327950288f
- #define M_PI_2_F 1.57079632679489661923132169163975144f
- #define M_PI_4_F 0.78539816339744830961566084581987572 1f
- #define M_1_PI_F 0.31830988618379067153776752674502872 4f
- #define M_2_PI_F 0.63661977236758134307553505349005744 8f
- #define M_2_SQRTPI_F 1.12837916709551257389615890312154517f
- #define M_SQRT2_F 1.41421356237309504880168872420969808f
- #define M_SQRT1_2_F 0.70710678118654752440084436210484903 9f
- #define M_E 0x1.5bf0a8b145769p+1
- #define M_LOG2E 0x1.71547652b82fep+0
- #define M_LOG10E 0x1.bcb7b1526e50ep-2
- #define M_LN2 0x1.62e42fefa39efp-1

- #define M_LN10 0x1.26bb1bbb55516p+1
- #define M_PI 0x1.921fb54442d18p+1
- #define M_PI_2 0x1.921fb54442d18p+0
- #define M_PI_4 0x1.921fb54442d18p-1
- #define M_1_PI 0x1.45f306dc9c883p-2
- #define M_2_PI 0x1.45f306dc9c883p-1
- #define M_2_SQRTPI 0x1.20dd750429b6dp+0
- #define M_SQRT2 0x1.6a09e667f3bcdp+0
- #define M_SQRT1_2 0x1.6a09e667f3bcdp-1
- #define CHAR_BIT 8
- #define SCHAR_MAX 127
- #define SCHAR_MIN (-128)
- #define UCHAR_MAX 255
- #define CHAR_MAX SCHAR_MAX
- #define CHAR_MIN SCHAR_MIN
- #define USHRT_MAX 65535
- #define SHRT_MAX 32767
- #define SHRT_MIN (-32768)
- #define UINT_MAX 0xffffffff
- #define INT_MAX 2147483647
- #define INT_MIN (-2147483647-1)
- #define ULONG_MAX 0xffffffffffffffffUL
- #define LONG_MAX ((long)0x7fffffffffffffffL)
- #define LONG_MIN ((long)(-0x7fffffffffffffffL-1))
- #define DBL_DIG 15
- #define DBL_MANT_DIG 53
- #define DBL_MAX_10_EXP +308
- #define DBL_MAX_EXP +1024
- #define DBL_MIN_10_EXP -307
- #define DBL_MIN_EXP -1021
- #define DBL_MAX 0x1.fffffffffffffp1023
- #define DBL_MIN 0x1.0p-1022
- #define DBL_EPSILON 0x1.0p-52
- #define CLK_LOCAL_MEM_FENCE 0x1

  Queue a memory fence to ensure correct ordering of memory operations to local memory.
- #define CLK_GLOBAL_MEM_FENCE 0x2

  Queue a memory fence to ensure correct ordering of memory operations to global memory.
- #define __kernel_exec(X, typen)
- #define kernel_exec(X, typen)
- #define CLK_ADDRESS_NONE 0

  Use the coordinate (x, y) to do an element lookup in the 2D image object specified by image.
- #define CLK_ADDRESS_CLAMP_TO_EDGE 2
- #define CLK_ADDRESS_CLAMP 4
- #define CLK_ADDRESS_REPEAT 6
- #define CLK_ADDRESS_MIRRORED_REPEAT 8
- #define CLK_NORMALIZED_COORDS_FALSE 0
- #define CLK_NORMALIZED_COORDS_TRUE 1
- #define CLK_FILTER_NEAREST 0x10
- #define CLK_FILTER_LINEAR 0x20
- #define CLK_R 0x10B0

  Return the channel data type.
- #define CLK_A 0x10B1
- #define CLK_RG 0x10B2
- #define CLK_RA 0x10B3

- #define CLK_RGB 0x10B4
- #define CLK_RGBA 0x10B5
- #define CLK_BGRA 0x10B6
- #define CLK_ARGB 0x10B7
- #define CLK_INTENSITY 0x10B8
- #define CLK_LUMINANCE 0x10B9
- #define CLK_Rx 0x10BA
- #define CLK_RGx 0x10BB
- #define CLK_RGBx 0x10BC
- #define CLK_DEPTH 0x10BD
- #define CLK_DEPTH_STENCIL 0x10BE
- #define CLK_SNORM_INT8 0x10D0
- #define CLK_SNORM_INT16 0x10D1
- #define CLK_UNORM_INT8 0x10D2
- #define CLK_UNORM_INT16 0x10D3
- #define CLK_UNORM_SHORT_565 0x10D4
- #define CLK_UNORM_SHORT_555 0x10D5
- #define CLK_UNORM_INT_101010 0x10D6
- #define CLK_SIGNED_INT8 0x10D7
- #define CLK_SIGNED_INT16 0x10D8
- #define CLK_SIGNED_INT32 0x10D9
- #define CLK_UNSIGNED_INT8 0x10DA
- #define CLK_UNSIGNED_INT16 0x10DB
- #define CLK_UNSIGNED_INT32 0x10DC
- #define CLK_HALF_FLOAT 0x10DD
- #define CLK_FLOAT 0x10DE
- #define CLK_UNORM_INT24 0x10DF
- #define as_char(x) __builtin_astype((x), char)

  OpenCL as_typen operators Reinterprets a data type as another data type of the same size.
- #define as_char2(x) __builtin_astype((x), char2)
- #define as_char3(x) __builtin_astype((x), char3)
- #define as_char4(x) __builtin_astype((x), char4)
- #define as_char8(x) __builtin_astype((x), char8)
- #define as_char16(x) __builtin_astype((x), char16)
- #define as_uchar(x) __builtin_astype((x), uchar)
- #define as_uchar2(x) __builtin_astype((x), uchar2)
- #define as_uchar3(x) __builtin_astype((x), uchar3)
- #define as_uchar4(x) __builtin_astype((x), uchar4)
- #define as_uchar8(x) __builtin_astype((x), uchar8)
- #define as_uchar16(x) __builtin_astype((x), uchar16)
- #define as_short(x) __builtin_astype((x), short)
- #define as_short2(x) __builtin_astype((x), short2)
- #define as_short3(x) __builtin_astype((x), short3)
- #define as_short4(x) __builtin_astype((x), short4)
- #define as_short8(x) __builtin_astype((x), short8)
- #define as_short16(x) __builtin_astype((x), short16)
- #define as_ushort(x) __builtin_astype((x), ushort)
- #define as_ushort2(x) __builtin_astype((x), ushort2)
- #define as_ushort3(x) __builtin_astype((x), ushort3)
- #define as_ushort4(x) __builtin_astype((x), ushort4)
- #define as_ushort8(x) __builtin_astype((x), ushort8)
- #define as_ushort16(x) __builtin_astype((x), ushort16)
- #define as_int(x) __builtin_astype((x), int)
- #define as_int2(x) __builtin_astype((x), int2)

- #define as_int3(x) __builtin_astype((x), int3)
- #define as_int4(x) __builtin_astype((x), int4)
- #define as_int8(x) __builtin_astype((x), int8)
- #define as_int16(x) __builtin_astype((x), int16)
- #define as_uint(x) __builtin_astype((x), uint)
- #define as_uint2(x) __builtin_astype((x), uint2)
- #define as_uint3(x) __builtin_astype((x), uint3)
- #define as_uint4(x) __builtin_astype((x), uint4)
- #define as_uint8(x) __builtin_astype((x), uint8)
- #define as_uint16(x) __builtin_astype((x), uint16)
- #define as_long(x) __builtin_astype((x), long)
- #define as_long2(x) __builtin_astype((x), long2)
- #define as_long3(x) __builtin_astype((x), long3)
- #define as_long4(x) __builtin_astype((x), long4)
- #define as_long8(x) __builtin_astype((x), long8)
- #define as_long16(x) __builtin_astype((x), long16)
- #define as_ulong(x) __builtin_astype((x), ulong)
- #define as_ulong2(x) __builtin_astype((x), ulong2)
- #define as_ulong3(x) __builtin_astype((x), ulong3)
- #define as_ulong4(x) __builtin_astype((x), ulong4)
- #define as_ulong8(x) __builtin_astype((x), ulong8)
- #define as_ulong16(x) __builtin_astype((x), ulong16)
- #define as_half(x) __builtin_astype((x), half)
- #define as_half2(x) __builtin_astype((x), half2)
- #define as_half3(x) __builtin_astype((x), half3)
- #define as_half4(x) __builtin_astype((x), half4)
- #define as_half8(x) __builtin_astype((x), half8)
- #define as_half16(x) __builtin_astype((x), half16)
- #define as_float(x) __builtin_astype((x), float)
- #define as_float2(x) __builtin_astype((x), float2)
- #define as_float3(x) __builtin_astype((x), float3)
- #define as_float4(x) __builtin_astype((x), float4)
- #define as_float8(x) __builtin_astype((x), float8)
- #define as_float16(x) __builtin_astype((x), float16)
- #define as_double(x) __builtin_astype((x), double)
- #define as_double2(x) __builtin_astype((x), double2)
- #define as_double3(x) __builtin_astype((x), double3)
- #define as_double4(x) __builtin_astype((x), double4)
- #define as_double8(x) __builtin_astype((x), double8)
- #define as_double16(x) __builtin_astype((x), double16)

Typedefs

- typedef unsigned char uchar

  An unsigned 8-bit integer.
- typedef unsigned short ushort

  An unsigned 16-bit integer.
- typedef unsigned int uint

  An unsigned 32-bit integer.
- typedef unsigned long ulong

  An unsigned 64-bit integer.
- typedef ptrdiff_t intptr_t

  The unsigned integer type of the result of the sizeof operator.

- typedef size_t uintptr_t
- typedef char char2 __attribute__((ext_vector_type(2)))

  Returns the number of global work-items specified for dimension identified by dimindx.
- typedef uint cl_mem_fence_flags

  All work-items in a work-group executing the kernel on a processor must execute this function before any are allowed to continue execution beyond the barrier.

Functions

- uint const_func __attribute__ ((overloadable)) get_work_dim(void)

  Returns the number of dimensions in use.
- int printf (__constant const char *st,...)

Variables

- float const_func float x
- float const_func float y
- float const_func float b
- float const_func float float c
- float __global float * iptr
- float __global int * exp
- float const_func int n
- float __global int * signp
- float float __global int * quo
- float __global float * cosval
- char const_func char minval
- char const_func char char maxval
- char const_func char i
- short const_func uchar lo
- int const_func int int z
- float const_func float float a
- float const_func float edge1
- float4 const_func float4 p1
- char2 const __global char * p
- void size_t offset
- event_t const __global char * src
- event_t const __global char size_t num_elements
- event_t const __global char size_t event_t event
- event_t const __global char size_t size_t src_stride
- event_t const __local char size_t size_t dst_stride
- void event_t * event_list
- int int val
- int int cmp
- char2 const_func uchar2 mask
- float4 sampler_t sampler
- float4 sampler_t int2 coord
- void int2 float4 color

11.63.1 Macro Definition Documentation

11.63.1.1 __kernel_exec

```
#define __kernel_exec(
              X,
              typen )
```

Value:

```
__kernel \
    __attribute__((work_group_size_hint(X, 1, 1))) \
    __attribute__((vec_type_hint(typen)))
```

Definition at line 6483 of file opencl_spir.h.

11.63.1.2 as_char

```
#define as_char(
              x ) __builtin_astype((x), char)
```

OpenCL as_typen operators Reinterprets a data type as another data type of the same size.

Definition at line 7497 of file opencl_spir.h.

11.63.1.3 as_char16

```
#define as_char16(
              x ) __builtin_astype((x), char16)
```

Definition at line 7502 of file opencl_spir.h.

11.63.1.4 as_char2

```
#define as_char2(
              x ) __builtin_astype((x), char2)
```

Definition at line 7498 of file opencl_spir.h.

11.63.1.5 as_char3

```
#define as_char3(
              x ) __builtin_astype((x), char3)
```

Definition at line 7499 of file opencl_spir.h.

11.63.1.6 as_char4

```
#define as_char4(
            x ) __builtin_astype((x), char4)
```

Definition at line 7500 of file opencl_spir.h.

11.63.1.7 as_char8

```
#define as_char8(
            x ) __builtin_astype((x), char8)
```

Definition at line 7501 of file opencl_spir.h.

11.63.1.8 as_double

```
#define as_double(
            x ) __builtin_astype((x), double)
```

Definition at line 7567 of file opencl_spir.h.

11.63.1.9 as_double16

```
#define as_double16(
            x ) __builtin_astype((x), double16)
```

Definition at line 7572 of file opencl_spir.h.

11.63.1.10 as_double2

```
#define as_double2(
            x ) __builtin_astype((x), double2)
```

Definition at line 7568 of file opencl_spir.h.

11.63.1.11 as_double3

```
#define as_double3(
            x ) __builtin_astype((x), double3)
```

Definition at line 7569 of file opencl_spir.h.

11.63.1.12 as_double4

```
#define as_double4(
            x ) __builtin_astype((x), double4)
```

Definition at line 7570 of file opencl_spir.h.

11.63.1.13 as_double8

```
#define as_double8(
            x ) __builtin_astype((x), double8)
```

Definition at line 7571 of file opencl_spir.h.

11.63.1.14 as_float

```
#define as_float(
            x ) __builtin_astype((x), float)
```

Definition at line 7560 of file opencl_spir.h.

11.63.1.15 as_float16

```
#define as_float16(
            x ) __builtin_astype((x), float16)
```

Definition at line 7565 of file opencl_spir.h.

11.63.1.16 as_float2

```
#define as_float2(
            x ) __builtin_astype((x), float2)
```

Definition at line 7561 of file opencl_spir.h.

11.63.1.17 as_float3

```
#define as_float3(
            x ) __builtin_astype((x), float3)
```

Definition at line 7562 of file opencl_spir.h.

11.63.1.18 as_float4

```
#define as_float4(
            x ) __builtin_astype((x), float4)
```

Definition at line 7563 of file opencl_spir.h.

11.63.1.19 as_float8

```
#define as_float8(
            x ) __builtin_astype((x), float8)
```

Definition at line 7564 of file opencl_spir.h.

11.63.1.20 as_half

```
#define as_half(
            x ) __builtin_astype((x), half)
```

Definition at line 7553 of file opencl_spir.h.

11.63.1.21 as_half16

```
#define as_half16(
            x ) __builtin_astype((x), half16)
```

Definition at line 7558 of file opencl_spir.h.

11.63.1.22 as_half2

```
#define as_half2(
            x ) __builtin_astype((x), half2)
```

Definition at line 7554 of file opencl_spir.h.

11.63.1.23 as_half3

```
#define as_half3(
            x ) __builtin_astype((x), half3)
```

Definition at line 7555 of file opencl_spir.h.

11.63.1.24 as_half4

```
#define as_half4(
            x ) __builtin_astype((x), half4)
```

Definition at line 7556 of file opencl_spir.h.

11.63.1.25 as_half8

```
#define as_half8(
            x ) __builtin_astype((x), half8)
```

Definition at line 7557 of file opencl_spir.h.

11.63.1.26 as_int

```
#define as_int(
            x ) __builtin_astype((x), int)
```

Definition at line 7525 of file opencl_spir.h.

11.63.1.27 as_int16

```
#define as_int16(
            x ) __builtin_astype((x), int16)
```

Definition at line 7530 of file opencl_spir.h.

11.63.1.28 as_int2

```
#define as_int2(
            x ) __builtin_astype((x), int2)
```

Definition at line 7526 of file opencl_spir.h.

11.63.1.29 as_int3

```
#define as_int3(
            x ) __builtin_astype((x), int3)
```

Definition at line 7527 of file opencl_spir.h.

11.63.1.30 as_int4

```
#define as_int4(
            x ) __builtin_astype((x), int4)
```

Definition at line 7528 of file opencl_spir.h.

11.63.1.31 as_int8

```
#define as_int8(
            x ) __builtin_astype((x), int8)
```

Definition at line 7529 of file opencl_spir.h.

11.63.1.32 as_long

```
#define as_long(
            x ) __builtin_astype((x), long)
```

Definition at line 7539 of file opencl_spir.h.

11.63.1.33 as_long16

```
#define as_long16(
            x ) __builtin_astype((x), long16)
```

Definition at line 7544 of file opencl_spir.h.

11.63.1.34 as_long2

```
#define as_long2(
            x ) __builtin_astype((x), long2)
```

Definition at line 7540 of file opencl_spir.h.

11.63.1.35 as_long3

```
#define as_long3(
            x ) __builtin_astype((x), long3)
```

Definition at line 7541 of file opencl_spir.h.

11.63.1.36 as_long4

```
#define as_long4(
            x ) __builtin_astype((x), long4)
```

Definition at line 7542 of file opencl_spir.h.

11.63.1.37 as_long8

```
#define as_long8(
            x ) __builtin_astype((x), long8)
```

Definition at line 7543 of file opencl_spir.h.

11.63.1.38 as_short

```
#define as_short(
            x ) __builtin_astype((x), short)
```

Definition at line 7511 of file opencl_spir.h.

11.63.1.39 as_short16

```
#define as_short16(
            x ) __builtin_astype((x), short16)
```

Definition at line 7516 of file opencl_spir.h.

11.63.1.40 as_short2

```
#define as_short2(
            x ) __builtin_astype((x), short2)
```

Definition at line 7512 of file opencl_spir.h.

11.63.1.41 as_short3

```
#define as_short3(
            x ) __builtin_astype((x), short3)
```

Definition at line 7513 of file opencl_spir.h.

11.63.1.42 as_short4

```
#define as_short4(
            x ) __builtin_astype((x), short4)
```

Definition at line 7514 of file opencl_spir.h.

11.63.1.43 as_short8

```
#define as_short8(
            x ) __builtin_astype((x), short8)
```

Definition at line 7515 of file opencl_spir.h.

11.63.1.44 as_uchar

```
#define as_uchar(
            x ) __builtin_astype((x), uchar)
```

Definition at line 7504 of file opencl_spir.h.

11.63.1.45 as_uchar16

```
#define as_uchar16(
            x ) __builtin_astype((x), uchar16)
```

Definition at line 7509 of file opencl_spir.h.

11.63.1.46 as_uchar2

```
#define as_uchar2(
            x ) __builtin_astype((x), uchar2)
```

Definition at line 7505 of file opencl_spir.h.

11.63.1.47 as_uchar3

```
#define as_uchar3(
            x ) __builtin_astype((x), uchar3)
```

Definition at line 7506 of file opencl_spir.h.

11.63.1.48 as_uchar4

```
#define as_uchar4(
            x ) __builtin_astype((x), uchar4)
```

Definition at line 7507 of file opencl_spir.h.

11.63.1.49 as_uchar8

```
#define as_uchar8(
            x ) __builtin_astype((x), uchar8)
```

Definition at line 7508 of file opencl_spir.h.

11.63.1.50 as_uint

```
#define as_uint(
            x ) __builtin_astype((x), uint)
```

Definition at line 7532 of file opencl_spir.h.

11.63.1.51 as_uint16

```
#define as_uint16(
            x ) __builtin_astype((x), uint16)
```

Definition at line 7537 of file opencl_spir.h.

11.63.1.52 as_uint2

```
#define as_uint2(
            x ) __builtin_astype((x), uint2)
```

Definition at line 7533 of file opencl_spir.h.

11.63.1.53 as_uint3

```
#define as_uint3(
            x ) __builtin_astype((x), uint3)
```

Definition at line 7534 of file opencl_spir.h.

11.63.1.54 as_uint4

```
#define as_uint4(
            x ) __builtin_astype((x), uint4)
```

Definition at line 7535 of file opencl_spir.h.

11.63.1.55 as_uint8

```
#define as_uint8(
            x ) __builtin_astype((x), uint8)
```

Definition at line 7536 of file opencl_spir.h.

11.63.1.56 as_ulong

```
#define as_ulong(
            x ) __builtin_astype((x), ulong)
```

Definition at line 7546 of file opencl_spir.h.

11.63.1.57 as_ulong16

```
#define as_ulong16(
            x ) __builtin_astype((x), ulong16)
```

Definition at line 7551 of file opencl_spir.h.

11.63.1.58 as_ulong2

```
#define as_ulong2(
            x ) __builtin_astype((x), ulong2)
```

Definition at line 7547 of file opencl_spir.h.

11.63.1.59 as_ulong3

```
#define as_ulong3(
            x ) __builtin_astype((x), ulong3)
```

Definition at line 7548 of file opencl_spir.h.

11.63.1.60 as_ulong4

```
#define as_ulong4(
            x ) __builtin_astype((x), ulong4)
```

Definition at line 7549 of file opencl_spir.h.

11.63.1.61 as_ulong8

```
#define as_ulong8(
            x ) __builtin_astype((x), ulong8)
```

Definition at line 7550 of file opencl_spir.h.

11.63.1.62 as_ushort

```
#define as_ushort(
            x ) __builtin_astype((x), ushort)
```

Definition at line 7518 of file opencl_spir.h.

11.63.1.63 as_ushort16

```
#define as_ushort16(
            x ) __builtin_astype((x), ushort16)
```

Definition at line 7523 of file opencl_spir.h.

11.63.1.64 as_ushort2

```
#define as_ushort2(
            x ) __builtin_astype((x), ushort2)
```

Definition at line 7519 of file opencl_spir.h.

11.63.1.65 as_ushort3

```
#define as_ushort3(
            x ) __builtin_astype((x), ushort3)
```

Definition at line 7520 of file opencl_spir.h.

11.63.1.66 as_ushort4

```
#define as_ushort4(
            x ) __builtin_astype((x), ushort4)
```

Definition at line 7521 of file opencl_spir.h.

11.63.1.67 as_ushort8

```
#define as_ushort8(
            x ) __builtin_astype((x), ushort8)
```

Definition at line 7522 of file opencl_spir.h.

11.63.1.68 CHAR_BIT

```
#define CHAR_BIT 8
```

Definition at line 218 of file opencl_spir.h.

11.63.1.69 CHAR_MAX

```
#define CHAR_MAX SCHAR_MAX
```

Definition at line 222 of file opencl_spir.h.

11.63.1.70 CHAR_MIN

```
#define CHAR_MIN SCHAR_MIN
```

Definition at line 223 of file opencl_spir.h.

11.63.1.71 CLK_A

```
#define CLK_A 0x10B1
```

Definition at line 7356 of file opencl_spir.h.

11.63.1.72 CLK_ADDRESS_CLAMP

```
#define CLK_ADDRESS_CLAMP 4
```

Definition at line 7011 of file opencl_spir.h.

11.63.1.73 CLK_ADDRESS_CLAMP_TO_EDGE

```
#define CLK_ADDRESS_CLAMP_TO_EDGE 2
```

Definition at line 7010 of file opencl_spir.h.

11.63.1.74 CLK_ADDRESS_MIRRORED_REPEAT

```
#define CLK_ADDRESS_MIRRORED_REPEAT 8
```

Definition at line 7013 of file opencl_spir.h.

11.63.1.75 CLK_ADDRESS_NONE

```
#define CLK_ADDRESS_NONE 0
```

Use the coordinate (x, y) to do an element lookup in the 2D image object specified by image.

read_imagef returns floating-point values in the range [0.0 . . . 1.0] for image objects created with image_channel↩_data_type set to one of the predefined packed formats or CL_UNORM_INT8, or CL_UNORM_INT16. read_imagef returns floating-point values in the range [-1.0 . . . 1.0] for image objects created with image_channel_data_type set to CL_SNORM_INT8, or CL_SNORM_INT16. read_imagef returns floating-point values for image objects created with image_channel_data_type set to CL_HALF_FLOAT or CL_FLOAT. The read_imagef calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CL↩K_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by read_imagef for image objects with image_channel_data_type values not specified in the description above are undefined.

Definition at line 7009 of file opencl_spir.h.

11.63.1.76 CLK_ADDRESS_REPEAT

```
#define CLK_ADDRESS_REPEAT 6
```

Definition at line 7012 of file opencl_spir.h.

11.63.1.77 CLK_ARGB

```
#define CLK_ARGB 0x10B7
```

Definition at line 7362 of file opencl_spir.h.

11.63.1.78 CLK_BGRA

```
#define CLK_BGRA 0x10B6
```

Definition at line 7361 of file opencl_spir.h.

11.63.1.79 CLK_DEPTH

```
#define CLK_DEPTH 0x10BD
```

Definition at line 7368 of file opencl_spir.h.

11.63.1.80 CLK_DEPTH_STENCIL

```
#define CLK_DEPTH_STENCIL 0x10BE
```

Definition at line 7369 of file opencl_spir.h.

11.63.1.81 CLK_FILTER_LINEAR

```
#define CLK_FILTER_LINEAR 0x20
```

Definition at line 7025 of file opencl_spir.h.

11.63.1.82 CLK_FILTER_NEAREST

```
#define CLK_FILTER_NEAREST 0x10
```

Definition at line 7024 of file opencl_spir.h.

11.63.1.83 CLK_FLOAT

```
#define CLK_FLOAT 0x10DE
```

Definition at line 7388 of file opencl_spir.h.

11.63.1.84 CLK_GLOBAL_MEM_FENCE

```
#define CLK_GLOBAL_MEM_FENCE 0x2
```

Queue a memory fence to ensure correct ordering of memory operations to global memory.

Definition at line 5831 of file opencl_spir.h.

11.63.1.85 CLK_HALF_FLOAT

```
#define CLK_HALF_FLOAT 0x10DD
```

Definition at line 7387 of file opencl_spir.h.

11.63.1.86 CLK_INTENSITY

```
#define CLK_INTENSITY 0x10B8
```

Definition at line 7363 of file opencl_spir.h.

11.63.1.87 CLK_LOCAL_MEM_FENCE

```
#define CLK_LOCAL_MEM_FENCE 0x1
```

Queue a memory fence to ensure correct ordering of memory operations to local memory.

Definition at line 5825 of file opencl_spir.h.

11.63.1.88 CLK_LUMINANCE

```
#define CLK_LUMINANCE 0x10B9
```

Definition at line 7364 of file opencl_spir.h.

11.63.1.89 CLK_NORMALIZED_COORDS_FALSE

```
#define CLK_NORMALIZED_COORDS_FALSE 0
```

Definition at line 7018 of file opencl_spir.h.

11.63.1.90 CLK_NORMALIZED_COORDS_TRUE

```
#define CLK_NORMALIZED_COORDS_TRUE 1
```

Definition at line 7019 of file opencl_spir.h.

11.63.1.91 CLK_R

```
#define CLK_R 0x10B0
```

Return the channel data type.

Valid values are: CLK_SNORM_INT8 CLK_SNORM_INT16 CLK_UNORM_INT8 CLK_UNORM_INT16 CLK_↩ UNORM_SHORT_565 CLK_UNORM_SHORT_555 CLK_UNORM_SHORT_101010 CLK_SIGNED_INT8 CLK_↩ SIGNED_INT16 CLK_SIGNED_INT32 CLK_UNSIGNED_INT8 CLK_UNSIGNED_INT16 CLK_UNSIGNED_INT32 CLK_HALF_FLOAT CLK_FLOAT

Definition at line 7355 of file opencl_spir.h.

11.63.1.92 CLK_RA

```
#define CLK_RA 0x10B3
```

Definition at line 7358 of file opencl_spir.h.

11.63.1.93 CLK_RG

```
#define CLK_RG 0x10B2
```

Definition at line 7357 of file opencl_spir.h.

11.63.1.94 CLK_RGB

```
#define CLK_RGB 0x10B4
```

Definition at line 7359 of file opencl_spir.h.

11.63.1.95 CLK_RGBA

```
#define CLK_RGBA 0x10B5
```

Definition at line 7360 of file opencl_spir.h.

11.63.1.96 CLK_RGBx

```
#define CLK_RGBx 0x10BC
```

Definition at line 7367 of file opencl_spir.h.

11.63.1.97 CLK_RGx

```
#define CLK_RGx 0x10BB
```

Definition at line 7366 of file opencl_spir.h.

11.63.1.98 CLK_Rx

```
#define CLK_Rx 0x10BA
```

Definition at line 7365 of file opencl_spir.h.

11.63.1.99 CLK_SIGNED_INT16

```
#define CLK_SIGNED_INT16 0x10D8
```

Definition at line 7382 of file opencl_spir.h.

11.63.1.100 CLK_SIGNED_INT32

```
#define CLK_SIGNED_INT32 0x10D9
```

Definition at line 7383 of file opencl_spir.h.

11.63.1.101 CLK_SIGNED_INT8

```
#define CLK_SIGNED_INT8 0x10D7
```

Definition at line 7381 of file opencl_spir.h.

11.63.1.102 CLK_SNORM_INT16

```
#define CLK_SNORM_INT16 0x10D1
```

Definition at line 7375 of file opencl_spir.h.

11.63.1.103 CLK_SNORM_INT8

```
#define CLK_SNORM_INT8 0x10D0
```

Definition at line 7374 of file opencl_spir.h.

11.63.1.104 CLK_UNORM_INT16

```
#define CLK_UNORM_INT16 0x10D3
```

Definition at line 7377 of file opencl_spir.h.

11.63.1.105 CLK_UNORM_INT24

```
#define CLK_UNORM_INT24 0x10DF
```

Definition at line 7389 of file opencl_spir.h.

11.63.1.106 CLK_UNORM_INT8

```
#define CLK_UNORM_INT8 0x10D2
```

Definition at line 7376 of file opencl_spir.h.

11.63.1.107 CLK_UNORM_INT_101010

```
#define CLK_UNORM_INT_101010 0x10D6
```

Definition at line 7380 of file opencl_spir.h.

11.63.1.108 CLK_UNORM_SHORT_555

```
#define CLK_UNORM_SHORT_555 0x10D5
```

Definition at line 7379 of file opencl_spir.h.

11.63.1.109 CLK_UNORM_SHORT_565

```
#define CLK_UNORM_SHORT_565 0x10D4
```

Definition at line 7378 of file opencl_spir.h.

11.63.1.110 CLK_UNSIGNED_INT16

```
#define CLK_UNSIGNED_INT16 0x10DB
```

Definition at line 7385 of file opencl_spir.h.

11.63.1.111 CLK_UNSIGNED_INT32

```
#define CLK_UNSIGNED_INT32 0x10DC
```

Definition at line 7386 of file opencl_spir.h.

11.63.1.112 CLK_UNSIGNED_INT8

```
#define CLK_UNSIGNED_INT8 0x10DA
```

Definition at line 7384 of file opencl_spir.h.

11.63.1.113 const_func

```
#define const_func __attribute__((const))
```

Added according to pocl/examples/example1-spir64/generate_spir.sh.

Todo avoid this hack

Definition at line 21 of file opencl_spir.h.

11.63.1.114 DBL_DIG [1/2]

```
#define DBL_DIG 15
```

Definition at line 234 of file opencl_spir.h.

11.63.1.115 DBL_DIG [2/2]

```
#define DBL_DIG 15
```

Definition at line 234 of file opencl_spir.h.

11.63.1.116 DBL_EPSILON [1/2]

```
#define DBL_EPSILON 0x1.0p-52
```

Definition at line 242 of file opencl_spir.h.

11.63.1.117 DBL_EPSILON [2/2]

```
#define DBL_EPSILON 0x1.0p-52
```

Definition at line 242 of file opencl_spir.h.

11.63.1.118 DBL_MANT_DIG [1/2]

```
#define DBL_MANT_DIG 53
```

Definition at line 235 of file opencl_spir.h.

11.63.1.119 DBL_MANT_DIG [2/2]

```
#define DBL_MANT_DIG 53
```

Definition at line 235 of file opencl_spir.h.

11.63.1.120 DBL_MAX [1/2]

```
#define DBL_MAX 0x1.fffffffffffffp1023
```

Definition at line 240 of file opencl_spir.h.

11.63.1.121 DBL_MAX [2/2]

```
#define DBL_MAX 0x1.fffffffffffffp1023
```

Definition at line 240 of file opencl_spir.h.

11.63.1.122 DBL_MAX_10_EXP [1/2]

```
#define DBL_MAX_10_EXP +308
```

Definition at line 236 of file opencl_spir.h.

11.63.1.123 DBL_MAX_10_EXP [2/2]

```
#define DBL_MAX_10_EXP +308
```

Definition at line 236 of file opencl_spir.h.

11.63.1.124 DBL_MAX_EXP [1/2]

```
#define DBL_MAX_EXP +1024
```

Definition at line 237 of file opencl_spir.h.

11.63.1.125 DBL_MAX_EXP [2/2]

```
#define DBL_MAX_EXP +1024
```

Definition at line 237 of file opencl_spir.h.

11.63.1.126 DBL_MIN [1/2]

```
#define DBL_MIN 0x1.0p-1022
```

Definition at line 241 of file opencl_spir.h.

11.63.1.127 DBL_MIN [2/2]

```
#define DBL_MIN 0x1.0p-1022
```

Definition at line 241 of file opencl_spir.h.

11.63.1.128 DBL_MIN_10_EXP [1/2]

```
#define DBL_MIN_10_EXP -307
```

Definition at line 238 of file opencl_spir.h.

11.63.1.129 DBL_MIN_10_EXP [2/2]

```
#define DBL_MIN_10_EXP -307
```

Definition at line 238 of file opencl_spir.h.

11.63.1.130 DBL_MIN_EXP [1/2]

```
#define DBL_MIN_EXP -1021
```

Definition at line 239 of file opencl_spir.h.

11.63.1.131 DBL_MIN_EXP [2/2]

```
#define DBL_MIN_EXP -1021
```

Definition at line 239 of file opencl_spir.h.

11.63.1.132 DBL_RADIX

```
#define DBL_RADIX 2
```

Definition at line 184 of file opencl_spir.h.

11.63.1.133 FLT_DIG

```
#define FLT_DIG 6
```

Definition at line 167 of file opencl_spir.h.

11.63.1.134 FLT_EPSILON

```
#define FLT_EPSILON 0x1.0p-23f
```

Definition at line 176 of file opencl_spir.h.

11.63.1.135 FLT_MANT_DIG

```
#define FLT_MANT_DIG 24
```

Definition at line 168 of file opencl_spir.h.

11.63.1.136 FLT_MAX

```
#define FLT_MAX 0x1.fffffep127f
```

Definition at line 174 of file opencl_spir.h.

11.63.1.137 FLT_MAX_10_EXP

```
#define FLT_MAX_10_EXP +38
```

Definition at line 169 of file opencl_spir.h.

11.63.1.138 FLT_MAX_EXP

```
#define FLT_MAX_EXP +128
```

Definition at line 170 of file opencl_spir.h.

11.63.1.139 FLT_MIN

```
#define FLT_MIN 0x1.0p-126f
```

Definition at line 175 of file opencl_spir.h.

11.63.1.140 FLT_MIN_10_EXP

```
#define FLT_MIN_10_EXP -37
```

Definition at line 171 of file opencl_spir.h.

11.63.1.141 FLT_MIN_EXP

```
#define FLT_MIN_EXP -125
```

Definition at line 172 of file opencl_spir.h.

11.63.1.142 FLT_RADIX

```
#define FLT_RADIX 2
```

Definition at line 173 of file opencl_spir.h.

11.63.1.143 FP_ILOGB0

```
#define FP_ILOGB0 INT_MIN
```

Definition at line 164 of file opencl_spir.h.

11.63.1.144 FP_ILOGBNAN

```
#define FP_ILOGBNAN INT_MAX
```

Definition at line 165 of file opencl_spir.h.

11.63.1.145 HUGE_VAL

```
#define HUGE_VAL (__builtin_huge_val())
```

A positive double constant expression.

HUGE_VAL evaluates to +infinity. Used as an error value returned by the built-in math functions.

Definition at line 151 of file opencl_spir.h.

11.63.1.146 HUGE_VALF

```
#define HUGE_VALF (__builtin_huge_valf())
```

A positive float constant expression.

HUGE_VALF evaluates to +infinity. Used as an error value returned by the built-in math functions.

Definition at line 144 of file opencl_spir.h.

11.63.1.147 INFINITY

```
#define INFINITY (__builtin_inff())
```

A constant expression of type float representing positive or unsigned infinity.

Definition at line 157 of file opencl_spir.h.

11.63.1.148 INT_MAX

```
#define INT_MAX 2147483647
```

Definition at line 228 of file opencl_spir.h.

11.63.1.149 INT_MIN

```
#define INT_MIN (-2147483647-1)
```

Definition at line 229 of file opencl_spir.h.

11.63.1.150 kernel_exec

```
#define kernel_exec(
              X,
              typen )
```

Value:

```
__kernel \
    __attribute__((work_group_size_hint(X, 1, 1))) \
    __attribute__((vec_type_hint(typen)))
```

Definition at line 6487 of file opencl_spir.h.

11.63.1.151 LONG_MAX

```
#define LONG_MAX ((long)0x7fffffffffffffffL)
```

Definition at line 231 of file opencl_spir.h.

11.63.1.152 LONG_MIN

```
#define LONG_MIN ((long)(-0x7fffffffffffffffL-1))
```

Definition at line 232 of file opencl_spir.h.

11.63.1.153 M_1_PI

```
#define M_1_PI 0x1.45f306dc9c883p-2
```

Definition at line 212 of file opencl_spir.h.

11.63.1.154 M_1_PI_F

```
#define M_1_PI_F 0.31830988618379067153776752674502872 4f
```

Definition at line 198 of file opencl_spir.h.

11.63.1.155 M_2_PI

```
#define M_2_PI 0x1.45f306dc9c883p-1
```

Definition at line 213 of file opencl_spir.h.

11.63.1.156 M_2_PI_F

```
#define M_2_PI_F 0.63661977236758134307553505349005744 8f
```

Definition at line 199 of file opencl_spir.h.

11.63.1.157 M_2_SQRTPI

```
#define M_2_SQRTPI 0x1.20dd750429b6dp+0
```

Definition at line 214 of file opencl_spir.h.

11.63.1.158 M_2_SQRTPI_F

```
#define M_2_SQRTPI_F 1.128379167095512573896158903121545 17f
```

Definition at line 200 of file opencl_spir.h.

11.63.1.159 M_E

```
#define M_E 0x1.5bf0a8b145769p+1
```

Definition at line 204 of file opencl_spir.h.

11.63.1.160 M_E_F

```
#define M_E_F 2.718281828459045235360287471352662 50f
```

Definition at line 190 of file opencl_spir.h.

11.63.1.161 M_LN10

```
#define M_LN10 0x1.26bb1bbb55516p+1
```

Definition at line 208 of file opencl_spir.h.

11.63.1.162 M_LN10_F

```
#define M_LN10_F 2.302585092994045684017991454684364 21f
```

Definition at line 194 of file opencl_spir.h.

11.63.1.163 M_LN2

```
#define M_LN2 0x1.62e42fefa39efp-1
```

Definition at line 207 of file opencl_spir.h.

11.63.1.164 M_LN2_F

```
#define M_LN2_F 0.69314718055994530941723212145817656 8f
```

Definition at line 193 of file opencl_spir.h.

11.63.1.165 M_LOG10E

```
#define M_LOG10E 0x1.bcb7b1526e50ep-2
```

Definition at line 206 of file opencl_spir.h.

11.63.1.166 M_LOG10E_F

```
#define M_LOG10E_F 0.43429448190325182765112891891660508 2f
```

Definition at line 192 of file opencl_spir.h.

11.63.1.167 M_LOG2E

```
#define M_LOG2E 0x1.71547652b82fep+0
```

Definition at line 205 of file opencl_spir.h.

11.63.1.168 M_LOG2E_F

```
#define M_LOG2E_F 1.4426950408889634073599246810018921 4f
```

Definition at line 191 of file opencl_spir.h.

11.63.1.169 M_PI

```
#define M_PI 0x1.921fb54442d18p+1
```

Definition at line 209 of file opencl_spir.h.

11.63.1.170 M_PI_2

```
#define M_PI_2 0x1.921fb54442d18p+0
```

Definition at line 210 of file opencl_spir.h.

11.63.1.171 M_PI_2_F

```
#define M_PI_2_F 1.5707963267948966192313216916397514 4f
```

Definition at line 196 of file opencl_spir.h.

11.63.1.172 M_PI_4

```
#define M_PI_4 0x1.921fb54442d18p-1
```

Definition at line 211 of file opencl_spir.h.

11.63.1.173 M_PI_4_F

```
#define M_PI_4_F 0.7853981633974483096156608458198757 21f
```

Definition at line 197 of file opencl_spir.h.

11.63.1.174 M_PI_F

```
#define M_PI_F 3.1415926535897932384626433832795028 8f
```

Definition at line 195 of file opencl_spir.h.

11.63.1.175 M_SQRT1_2

```
#define M_SQRT1_2 0x1.6a09e667f3bcdp-1
```

Definition at line 216 of file opencl_spir.h.

11.63.1.176 M_SQRT1_2_F

```
#define M_SQRT1_2_F 0.70710678118654752440084436210484903 9f
```

Definition at line 202 of file opencl_spir.h.

11.63.1.177 M_SQRT2

```
#define M_SQRT2 0x1.6a09e667f3bcdp+0
```

Definition at line 215 of file opencl_spir.h.

11.63.1.178 M_SQRT2_F

```
#define M_SQRT2_F 1.4142135623730950488016887242096980 8f
```

Definition at line 201 of file opencl_spir.h.

11.63.1.179 MAXFLOAT

```
#define MAXFLOAT 0x1.fffffep127f
```

Value of maximum non-infinite single-precision floating-point number.

Definition at line 137 of file opencl_spir.h.

11.63.1.180 NAN

```
#define NAN as_float(__builtin_nanf((char const *)""))
```

A constant expression of type float representing a quiet NaN.

Definition at line 162 of file opencl_spir.h.

11.63.1.181 readonly

```
#define readonly __attribute__((pure))
```

Definition at line 22 of file opencl_spir.h.

11.63.1.182 SCHAR_MAX

```
#define SCHAR_MAX 127
```

Definition at line 219 of file opencl_spir.h.

11.63.1.183 SCHAR_MIN

```
#define SCHAR_MIN (-128)
```

Definition at line 220 of file opencl_spir.h.

11.63.1.184 SHRT_MAX

```
#define SHRT_MAX 32767
```

Definition at line 225 of file opencl_spir.h.

11.63.1.185 SHRT_MIN

```
#define SHRT_MIN (-32768)
```

Definition at line 226 of file opencl_spir.h.

11.63.1.186 UCHAR_MAX

```
#define UCHAR_MAX 255
```

Definition at line 221 of file opencl_spir.h.

11.63.1.187 UINT_MAX

```
#define UINT_MAX 0xffffffff
```

Definition at line 227 of file opencl_spir.h.

11.63.1.188 ULONG_MAX

```
#define ULONG_MAX 0xffffffffffffffffUL
```

Definition at line 230 of file opencl_spir.h.

11.63.1.189 USHRT_MAX

```
#define USHRT_MAX 65535
```

Definition at line 224 of file opencl_spir.h.

11.63.2 Typedef Documentation

11.63.2.1 __attribute__

```
double16 const_func __attribute__ (
            (ext_vector_type(2))  )
```

Returns the number of global work-items specified for dimension identified by dimindx.

Write color value to location specified by coord.xy in the 2D image layer identified by index coord.z in the 2D image array.

Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.

Return the number of images in the 1D image array.

Return the number of images in the 2D image array.

Return the 3D image width, height, and depth as an int4 type.

Return the 2D image width and height as an int2 type.

Return the image channel order.

Return the image depth in pixels.

Return the image height in pixels.

Return the image width in pixels.

Write color value to location specified by coordinate (x, y, z) in the 3D image object specified by image.

Use the coordinate (coord.x, coord.y, coord.z) to do an element lookup in the 3D image object specified by image.

Write color value to location specified by coordinate (x, y) in the 2D image object specified by image.

Use the coordinate (x, y) to do an element lookup in the 2D image object specified by image.

The shuffle and shuffle2 built-in functions construct a permutation of elements from one or two input vectors respectively that are of the same type, returning a vector with the same element type as the input and length that is the same as the shuffle mask.

Swaps the old value stored at location p with new value given by val.

unsigned int atomic_sub (volatile __global unsigned int ∗p, unsigned int val) int atomic_sub (volatile __local int ∗p, int val) unsigned int atomic_sub (volatile __local unsigned int ∗p, unsigned int val) Read the 32-bit value (referred to as old) stored at location pointed by p.

Read the 32-bit value (referred to as old) stored at location pointed by p.

Prefetch num_elements ∗ sizeof(gentype) bytes into the global cache.

Wait for events that identify the async_work_group_copy operations to complete.

Perform an async gather of num_elements gentype elements from src to dst.

event_t async_work_group_copy (__global gentype ∗dst, const __local gentype ∗src, size_t num_elements, event_t event) Perform an async copy of num_elements gentype elements from src to dst.

Write memory barrier that orders only stores.

Read memory barrier that orders only loads.

Orders loads and stores of a work-item executing a kernel.

For n = 1, 2, 4, 8 and 16 read sizeof (halfn) bytes of data from address (p + (offset ∗ n)).

The floatn value given by data is converted to a halfn value using the appropriate rounding mode.

The float value given by data is first converted to a half value using the appropriate rounding mode.

Read sizeof (halfn) bytes of data from address (p + (offset ∗ n)).

Read sizeof (half) bytes of data from address (p + offset).

Write sizeof (gentypen) bytes given by data to address (p + (offset ∗ n)).

Return sizeof (gentypen) bytes of data read from address (p + (offset ∗ n)).

For each component of a vector type, result[i] = if MSB of c[i] is set ? b[i] : a[i].

Each bit of the result is the corresponding bit of a if the corresponding bit of c is 0.

Returns 1 if the most significant bit in all components of x is set; otherwise returns 0.

Returns 1 if the most significant bit in any component of x is set; otherwise returns 0.

Test for sign bit.

Test if arguments are unordered.

Test if arguments are ordered.

Test for a normal value.

Test for a NaN.

Test for infinity value (+ve or –ve) .

Test for finite value.

Returns the component-wise compare of (x < y) || (x > y) .

Returns the component-wise compare of x <= y.

Returns the component-wise compare of x < y.

Returns the component-wise compare of x >= y.

Returns the component-wise compare of x > y.

Returns the component-wise compare of x != y.

intn isequal (floatn x, floatn y) Returns the component-wise compare of x == y.

Returns the length of vector p computed as: half_sqrt(p.x2 + p.y2 + ...)

Returns fast_length(p0 – p1).

Returns a vector in the same direction as p but with a length of 1.

Return the length of vector p, i.e., sqrt(p.x2 + p.y 2 + ...)

Returns the distance between p0 and p1.

Compute dot product.

Returns the cross product of p0.xyz and p1.xyz.

Returns 1.0 if x > 0, -0.0 if x = -0.0, +0.0 if x = +0.0, or –1.0 if x < 0.

Returns 0.0 if x <= edge0 and 1.0 if x >= edge1 and performs smooth Hermite interpolation between 0 and 1when edge0 < x < edge1.

Returns 0.0 if x < edge, otherwise it returns 1.0.

Converts degrees to radians, i.e.

Returns the linear blend of x & y implemented as: x + (y – x) ∗ a a must be a value in the range 0.0 . . . 1.0.

Converts radians to degrees, i.e.

Returns fmin(fmax(x, minval), maxval).

Multiply two 24-bit integer values x and y.

Multiply two 24-bit integer values x and y and add the 32-bit integer result to the 32-bit integer z.

result[i] = ((long)hi[i] << 32) | lo[i] result[i] = ((ulong)hi[i] << 32) | lo[i]

result[i] = ((int)hi[i] << 16) | lo[i] result[i] = ((uint)hi[i] << 16) | lo[i]

result[i] = ((short)hi[i] << 8) | lo[i] result[i] = ((ushort)hi[i] << 8) | lo[i]

Returns x - y and saturates the result.

For each element in v, the bits are shifted left by the number of bits given by the corresponding element in i (subject to usual shift modulo rules described in section 6.3).

Computes x ∗ y and returns the high half of the product of x and y.

Returns a ∗ b + c and saturates the result.

Returns mul_hi(a, b) + c.

Returns the number of leading 0-bits in x, starting at the most significant bit position.

Returns min(max(x, minval), maxval).

Returns (x + y + 1) >> 1.

Returns (x + y) >> 1.

Returns x + y and saturates the result.

Returns | x – y | without modulo overflow.

Returns | x |.

Compute tangent over an implementation-defined range.

Compute square root over an implementation-defined range.

Compute sine over an implementation-defined range.

Compute inverse square root over an implementationdefined range.

Compute reciprocal over an implementation-defined range.

Compute a base 10 logarithm over an implementationdefined range.

Compute a base 2 logarithm over an implementationdefined range.

Compute natural logarithm over an implementationdefined range.

Compute the base- 10 exponential of x over an implementation-defined range.

Compute the base- 2 exponential of x over an implementation-defined range.

Compute the base- e exponential of x over an implementation-defined range.

Compute x / y over an implementation-defined range.

Compute cosine over an implementation-defined range.

Compute reciprocal.

Compute the base- 10 exponential of x.

Compute the base- 2 exponential of x.

Compute x / y.

Round to integral value using the round to zero rounding mode.

Compute the gamma function.

Compute tan (PI $*$ x).

Compute hyperbolic tangent.

Compute tangent.

Compute square root.

Compute sin (PI $*$ x).

Compute hyperbolic sine.

Compute sine and cosine of x.

Compute sine.

Compute inverse square root.

Return the integral value nearest to x rounding halfway cases away from zero, regardless of the current rounding direction.

Compute x to the power 1/y.

Round to integral value (using round to nearest even rounding mode) in floating-point format.

The remquo function computes the value r such that r = x - n$*$y, where n is the integer nearest the exact value of x/y.

Compute the value r such that r = x - n$*$y, where n is the integer nearest the exact value of x/y.

Compute x to the power y, where x is $>$= 0.

Compute x to the power y, where y is an integer.

Compute x to the power y.

Computes the next representable single-precision floating-point value following x in the direction of y.

Returns a quiet NaN.

Decompose a floating-point number.

Returns x if | x | $<$ | y |, y if | y | $<$ | x |, otherwise fmin(x, y).

Returns x if | x | $>$ | y |, y if | y | $>$ | x |, otherwise fmax(x, y).

mad approximates a $*$ b + c.

Compute the exponent of x, which is the integral part of logr | x |.

Compute a base e logarithm of (1.0 + x).

Compute a base 10 logarithm.

Compute a base 2 logarithm.

Compute natural logarithm.

Log gamma function.

Multiply x by 2 to the power n.

Return the exponent as an integer value.

Compute the value of the square root of x^2+ y^2 without undue overflow or underflow.

Extract mantissa and exponent from x.

Returns fmin(x – floor (x), 0x1.fffffep-1f).

Modulus.

Returns y if y < x, otherwise it returns x.

Returns y if x < y, otherwise it returns x.

Returns the correctly rounded floating-point representation of the sum of c with the infinitely precise product of a and b.

Round to integral value using the round to –ve infinity rounding mode.

x - y if x > y, +0 if x is less than or equal to y.

Compute absolute value of a floating-point number.

Compute e^x- 1.0.

Exponential base 10 function.

Exponential base 2 function.

Compute the base- e exponential of x.

Error function encountered in integrating the normal distribution.

Complementary error function.

Compute cos (PI ∗ x).

Compute hyperbolic consine.

Compute cosine.

Returns x with its sign changed to match the sign of y.

Round to integral value using the round to +ve infinity rounding mode.

Compute cube-root.

Compute atan2 (y, x) / PI.

Compute atan (x) / PI.

Hyperbolic arc tangent.

Arc tangent of y / x.

Arc tangent function.

Compute asin (x) / PI.

Inverse hyperbolic sine.

Arc sine function.

Compute acos (x) / PI.

Inverse hyperbolic cosine.

Arc cosine function.

get_global_offset returns the offset values specified in global_work_offset argument to clEnqueueNDRangeKernel.

get_group_id returns the work-group ID which is a number from 0 .

Returns the number of work-groups that will execute a kernel for dimension identified by dimindx.

Returns the unique local work-item ID i.e.

Returns the number of local work-items specified in dimension identified by dimindx.

Returns the unique global work-item ID value for dimension identified by dimindx.

This value is given by the global_work_size argument to clEnqueueNDRangeKernel. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_global_size() returns 1. For clEnqueueTask, this always returns 1.

The global work-item ID specifies the work-item ID based on the number of global work-items specified to execute the kernel. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_global_id() returns 0. For clEnqueueTask, this returns 0.

This value is given by the local_work_size argument to clEnqueueNDRangeKernel if local_work_size is not NU↩LL; otherwise the OpenCL implementation chooses an appropriate local_work_size value which is returned by this function. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_local_size() returns 1. For clEnqueueTask, this always returns 1.

a work-item within a specific work-group for dimension identified by dimindx. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_local_id() returns 0. For clEnqueueTask, this returns 0.

Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_num_groups () returns 1. For clEnqueueTask, this always returns 1.

. get_num_groups(dimindx) – 1. Valid values of dimindx are 0 to get_work_dim() – 1. For other values, get_group↩_id() returns 0. For clEnqueueTask, this returns 0.

Valid values of dimindx are 0 to get_work_dim() – 1. For other values, get_global_offset() returns 0. For cl↩EnqueueTask, this returns 0.

Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.

If one argument is a NaN, fmax() returns the other argument. If both arguments are NaNs, fmax() returns a NaN.

If one argument is a NaN, fmin() returns the other argument. If both arguments are NaNs, fmin() returns a NaN.

Returns x – y ∗ trunc (x/y).

floor(x) is returned in iptr.

For each component the mantissa returned is a float with magnitude in the interval [1/2, 1) or 0. Each component of x equals mantissa returned ∗ 2^exp.

Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the signp argument of lgamma_r.

Whether or how the product of a ∗ b is rounded and how supernormal or subnormal intermediate products are handled is not defined. mad is intended to be used where speed is preferred over accuracy.

The modf function breaks the argument x into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by iptr.

The nancode may be placed in the significand of the resulting NaN.

Thus, if y is less than x, nextafter() returns the largest representable floating-point number less than x.

If there are two integers closest to x/y, n shall be the even one. If r is zero, it is given the same sign as x.

If there are two integers closest to x/y, n shall be the even one. If r is zero, it is given the same sign as x. This is the same value that is returned by the remainder function. remquo also calculates the lower seven bits of the integral quotient x/y, and gives that value the same sign as x/y. It stores this signed value in the object pointed to by quo.

Refer to section 7.1 for description of rounding modes.

The computed sine is the return value and computed cosine is returned in cosval.

x must be in the range -2^16 ... +2^16.

x must be in the range -2^16 . . . +2^16.

x must be in the range -216 . . . +216.

The maximum error is implementation-defined.

The maximum error is implementation defined.

The maximum error is implementationdefined.

The range of x and y are implementation-defined. The maximum error is implementation-defined.

The intermediate sum does not modulo overflow.

Results are undefined if minval > maxval.

Bits shifted off the left side of the element are shifted back in from the right.

Refer to definition of mul24 to see how the 24-bit integer multiplication is performed.

x and y are 32-bit integers but only the low 24-bits are used to perform the multiplication. mul24 should only be used when values in x and y are in the range [- 2^23, 2^23-1] if x and y are signed integers and in the range [0, 2^24-1] if x and y are unsigned integers. If x and y are not in this range, the multiplication result is implementation-defined.

(180 / PI) ∗ radians.

If x and y are infinite or NaN, the return values are undefined.

If a is not in the range 0.0 . . . 1.0, the return values are undefined.

(PI / 180) ∗ degrees.

This is useful in cases where you would want a threshold function with a smooth transition. This is equivalent to: gentype t; t = clamp ((x – edge0) / (edge1 – edge0), 0, 1); return t ∗ t ∗ (3 – 2 ∗ t); Results are undefined if edge0 >= edge1 or if x, edge0 or edge1 is a NaN.

Returns 0.0 if x is a NaN.

The w component of float4 result returned will be 0.0.

This is calculated as length(p0 – p1).

fast_normalize is computed as: p ∗ half_rsqrt (p.x^2 + p.y^2 + . . .) The result shall be within 8192 ulps error from the infinitely precise result of if (all(p == 0.0f)) result = p; else result = p / sqrt (p.x^2 + p.y^2 + ...); with the following exceptions: 1) If the sum of squares is greater than FLT_MAX then the value of the floating-point values in the result vector are undefined. 2) If the sum of squares is less than FLT_MIN then the implementation may return back p. 3) If the device is in “denorms are flushed to zero” mode, individual operand elements with magnitude less than sqrt(FLT_MIN) may be flushed to zero before proceeding with the calculation.

isordered() takes arguments x and y, and returns the result isequal(x, x) && isequal(y, y).

isunordered() takes arguments x and y, returning non-zero if x or y is NaN, and zero otherwise.

The scalar version of the function returns a 1 if the sign bit in the float is set else returns 0. The vector version of the function returns the following for each component in floatn: a -1 if the sign bit in the float is set else returns 0.

Otherwise it is the corresponding bit of b.

For a scalar type, result = c ? b : a.

The address computed as (p + (offset ∗ n)) must be 8-bit aligned if gentype is char, uchar; 16-bit aligned if gentype is short, ushort; 32-bit aligned if gentype is int, uint, float; 64-bit aligned if gentype is long, ulong.

The data read is interpreted as a half value. The half value is converted to a float value and the float value is returned. The read address computed as (p + offset) must be 16-bit aligned.

The data read is interpreted as a halfn value. The halfn value read is converted to a floatn value and the floatn value is returned. The read address computed as (p + (offset ∗ n)) must be 16-bit aligned.

The half value is then written to address computed as (p + offset). The address computed as (p + offset) must be 16- bit aligned. vstore_half use the current rounding mode. The default current rounding mode is round to nearest even.

The halfn value is then written to address computed as (p + (offset ∗ n)). The address computed as (p + (offset ∗ n)) must be 16-bit aligned. vstore_halfn uses the current rounding mode. The default current rounding mode is round to nearest even.

The data read is interpreted as a halfn value. The halfn value read is converted to a floatn value and the floatn value is returned. The address computed as (p + (offset ∗ n)) must be aligned to sizeof (halfn) bytes. For n = 3, vloada_half3 reads a half3 from address (p + (offset ∗ 4)) and returns a float3. The address computed as (p + (offset ∗ 4)) must be aligned to sizeof (half) ∗ 4 bytes.

For n = 1, 2, 4, 8 and 16, the halfn value is written to the address computed as (p + (offset

- n)). The address computed as (p + (offset ∗ n)) must be aligned to sizeof (halfn) bytes. For n = 3, the half3 value is written to the address computed as (p + (offset ∗ 4)). The address computed as (p + (offset ∗ 4)) must be aligned to sizeof (half) ∗ 4 bytes. vstorea_halfn uses the current rounding mode. The default current rounding mode is round to nearest even.

  This means that loads and stores preceding the mem_fence will be committed to memory before any loads and stores following the mem_fence. The flags argument specifies the memory address space and can be set to a combination of the following literal values: CLK_LOCAL_MEM_FENCE CLK_GLOBAL_MEM_FENCE.

The flags argument specifies the memory address space and can be set to to a combination of the following literal values: CLK_LOCAL_MEM_FENCE CLK_GLOBAL_MEM_FENCE.

The async copy is performed by all work-items in a workgroup and this built-in function must therefore be encountered by all work-items in a workgroup executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by wait_group_events to wait for the async copy to finish. The event argument can also be used to associate the async_work_group_copy with a previous async copy allowing an event to be shared by multiple async copies; otherwise event should be zero. If event argument is non-zero, the event object supplied in event argument will be returned. This function does not perform any implicit synchronization of source data such as using a barrier before performing the copy.

The src_stride is the stride in elements for each gentype element read from src. The dst_stride is the stride in elements for each gentype element written to dst. The async gather is performed by all work-items in a work-group. This built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by wait_group_events to wait for the async copy to finish. The event argument can also be used to associate the async_work_group_strided_copy with a previous async copy allowing an event to be shared by multiple async copies; otherwise event should be zero. If event argument is non-zero, the event object supplied in event argument will be returned. This function does not perform any implicit synchronization of source data such as using a barrier before performing the copy.

The event objects specified in event_list will be released after the wait is performed. This function must be encountered by all workitems in a work-group executing the kernel with the same num_events and event objects specified in event_list; otherwise the results are undefined.

The prefetch instruction is applied to a work-item in a workgroup and does not affect the functional behavior of the kernel.

Compute (old + val) and store result at location pointed by p. The function returns old.

Compute (old - val) and store result at location pointed by p. The function returns old.

Returns old value.

Compute (old + 1) and store result at location pointed by p. The function returns old.

Compute (old - 1) and store result at location pointed by p. The function returns old.

Compute (old == cmp) ? val : old and store result at location pointed by p. The function returns old.

Compute min(old, val) and store minimum value at location pointed by p. The function returns old.

Compute max(old, val) and store maximum value at location pointed by p. The function returns old.

Compute (old & val) and store result at location pointed by p. The function returns old.

Compute (old | val) and store result at location pointed by p. The function returns old.

Compute (old ^ val) and store result at location pointed by p. The function returns old.

The size of each element in the mask must match the size of each element in the result. For shuffle, only the ilogb(2m-1) least significant bits of each mask element are considered. For shuffle2, only the ilogb(2m-1)+1 least significant bits of each mask element are considered. Other bits in the mask shall be ignored. The elements of the input vectors are numbered from left to right across one or both of the vectors. For this purpose, the number of elements in a vector is given by vec_step(gentypem). The shuffle mask operand specifies, for each element of the result vector, which element of the one or two input vectors the result element gets. Examples: uint4 mask = (uint4)(3, 2, 1, 0); float4 a; float4 r = shuffle(a, mask); // r.s0123 = a.wzyx uint8 mask = (uint8)(0, 1, 2, 3, 4, 5, 6, 7); float4 a, b; float8 r = shuffle2(a, b, mask); // r.s0123 = a.xyzw // r.s4567 = b.xyzw uint4 mask; float8 a; float4 b; b = shuffle(a, mask); Examples that are not valid are: uint8 mask; short16 a; short8 b; b = shuffle(a, mask); <- not valid

read_imagei and read_imageui return unnormalized signed integer and unsigned integer values respectively. Each channel will be stored in a 32-bit integer. read_imagei can only be used with image objects created with image↩_channel_data_type set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGN↩ED_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imagei are undefined. read_imageui can only be used with image objects created with image_channel_data_type set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNED_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imageui are undefined. The read_image{i|ui} calls support a nearest filter only. The filter_mode specified in sampler must be set to CLK_FI↩LTER_NEAREST; otherwise the values returned are undefined. Furthermore, the read_image{i|ui} calls that take integer coordinates must use a sampler with normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRES↩S_NONE; otherwise the values returned are undefined.

Appropriate data format conversion to the specified image format is done before writing the color value. x & y are considered to be unnormalized coordinates and must be in the range 0 ... image width - 1, and 0 ... image height - 1. write_imagef can only be used with image objects created with image_channel_data_type set to one of the pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_↩INT16, CL_HALF_FLOAT or CL_FLOAT. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. write_imagei can only be used with image objects created with image_channel_data_type set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGNED_INT32. write_imageui can only be used with image objects created with image_channel_data_type set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNED_INT32. The behavior of write_imagef, write_imagei and write_imageui for image objects created with image_channel_data_type values not specified in the description above or with (x, y) coordinate values that are not in the range (0 ... image width - 1, 0 ... image height - 1), respectively, is undefined.

coord.w is ignored. read_imagef returns floating-point values in the range [0.0 ... 1.0] for image objects created with image_channel_data_type set to one of the predefined packed formats or CL_UNORM_INT8, or CL_UNORM_I↩NT16. read_imagef returns floating-point values in the range [-1.0 ... 1.0] for image objects created with image↩_channel_data_type set to CL_SNORM_INT8, or CL_SNORM_INT16. read_imagef returns floating-point values for image objects created with image_channel_data_type set to CL_HALF_FLOAT or CL_FLOAT. The read_↩imagef calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDR↩ESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by read_imagef for image objects with image_channel_data_type values not specified in the description are undefined.

coord.w is ignored. read_imagei and read_imageui return unnormalized signed integer and unsigned integer values respectively. Each channel will be stored in a 32-bit integer. read_imagei can only be used with image objects created with image_channel_data_type set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGNED_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imagei are undefined. read_imageui can only be used with image objects created with image_channel_↩data_type set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNE↩D_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imageui are undefined. The read_image{i|ui} calls support a nearest filter only. The filter_mode specified in sampler must be set to CLK_FILTER_NEAREST; otherwise the values returned are undefined. Furthermore, the read_image{i|ui} calls that take integer coordinates must use a sampler with normalized coordinates set to CLK_NORMALIZED_↩COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined.

Valid values are: CLK_A CLK_R CLK_Rx CLK_RG CLK_RGx CLK_RA CLK_RGB CLK_RGBx CLK_RGBA CLK↩_ARGB CLK_BGRA CLK_INTENSITY CLK_LUMINANCE

The width is returned in the x component, and the height in the y component.

The width is returned in the x component, height in the y component, depth in the z component and the w component is 0.

Definition at line 77 of file opencl_spir.h.

11.63.2.2 cl_mem_fence_flags

```
typedef uint cl_mem_fence_flags
```

All work-items in a work-group executing the kernel on a processor must execute this function before any are allowed to continue execution beyond the barrier.

This function must be encountered by all work-items in a work-group executing the kernel. If barrier is inside a conditional statement, then all work-items must enter the conditional if any work-item enters the conditional statement and executes the barrier. If barrer is inside a loop, all work-items must execute the barrier for each iteration of the loop before any are allowed to continue execution beyond the barrier. The barrier function also queues a memory fence (reads and writes) to ensure correct ordering of memory operations to local or global memory. The flags argument specifies the memory address space and can be set to a combination of the following literal values. CLK_LOCAL_MEM_FENCE - The barrier function will either flush any variables stored in local memory or queue a memory fence to ensure correct ordering of memory operations to local memory. CLK_GLOBAL_MEM_FENCE – The barrier function will queue a memory fence to ensure correct ordering of memory operations to global memory. This can be useful when work-items, for example, write to buffer or image objects and then want to read the updated data.

Definition at line 5774 of file opencl_spir.h.

11.63.2.3 intptr_t

```
typedef ptrdiff_t intptr_t
```

The unsigned integer type of the result of the sizeof operator.

This is a 32-bit unsigned integer if CL_DEVICE_ADDRESS_BITS defined in table 4.3 is 32-bits and is a 64-bit unsigned integer if CL_DEVICE_ADDRESS_BITS is 64-bits. A signed integer type that is the result of subtracting two pointers. This is a 32-bit signed integer if CL_DEVICE_ADDRESS_BITS defined in table 4.3 is 32-bits and is a 64-bit signed integer if CL_DEVICE_ADDRESS_BITS is 64-bits.

Definition at line 70 of file opencl_spir.h.

11.63.2.4 uchar

```
typedef unsigned char uchar
```

An unsigned 8-bit integer.

Definition at line 29 of file opencl_spir.h.

11.63.2.5 uint

```
typedef unsigned int uint
```

An unsigned 32-bit integer.

Definition at line 39 of file opencl_spir.h.

11.63.2.6 uintptr_t

```
typedef size_t uintptr_t
```

Definition at line 72 of file opencl_spir.h.

11.63.2.7 ulong

```
typedef unsigned long ulong
```

An unsigned 64-bit integer.

Definition at line 44 of file opencl_spir.h.

11.63.2.8 ushort

```
typedef unsigned short ushort
```

An unsigned 16-bit integer.

Definition at line 34 of file opencl_spir.h.

11.63.3 Function Documentation

11.63.3.1 __attribute__()

```
uint const_func __attribute__ (
            (overloadable)  )
```

Returns the number of dimensions in use.

Write color value to location specified by coord.xy in the 2D image layer identified by index coord.z in the 2D image array.

Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.

Return the number of images in the 1D image array.

Return the number of images in the 2D image array.

Return the 3D image width, height, and depth as an int4 type.

Return the 2D image width and height as an int2 type.

Return the image channel order.

Return the image depth in pixels.

Return the image height in pixels.

Return the image width in pixels.

Write color value to location specified by coordinate (x, y, z) in the 3D image object specified by image.

Use the coordinate (coord.x, coord.y, coord.z) to do an element lookup in the 3D image object specified by image.

Write color value to location specified by coordinate (x, y) in the 2D image object specified by image.

Use the coordinate (x, y) to do an element lookup in the 2D image object specified by image.

The shuffle and shuffle2 built-in functions construct a permutation of elements from one or two input vectors respectively that are of the same type, returning a vector with the same element type as the input and length that is the same as the shuffle mask.

Swaps the old value stored at location p with new value given by val.

unsigned int atomic_sub (volatile __global unsigned int ∗p, unsigned int val) int atomic_sub (volatile __local int ∗p, int val) unsigned int atomic_sub (volatile __local unsigned int ∗p, unsigned int val) Read the 32-bit value (referred to as old) stored at location pointed by p.

Read the 32-bit value (referred to as old) stored at location pointed by p.

Prefetch num_elements ∗ sizeof(gentype) bytes into the global cache.

Wait for events that identify the async_work_group_copy operations to complete.

Perform an async gather of num_elements gentype elements from src to dst.

event_t async_work_group_copy (__global gentype ∗dst, const __local gentype ∗src, size_t num_elements, event_t event) Perform an async copy of num_elements gentype elements from src to dst.

Write memory barrier that orders only stores.

Read memory barrier that orders only loads.

Orders loads and stores of a work-item executing a kernel.

For n = 1, 2, 4, 8 and 16 read sizeof (halfn) bytes of data from address (p + (offset $*$ n)).

The floatn value given by data is converted to a halfn value using the appropriate rounding mode.

The float value given by data is first converted to a half value using the appropriate rounding mode.

Read sizeof (halfn) bytes of data from address (p + (offset $*$ n)).

Read sizeof (half) bytes of data from address (p + offset).

Write sizeof (gentypen) bytes given by data to address (p + (offset $*$ n)).

Return sizeof (gentypen) bytes of data read from address (p + (offset $*$ n)).

For each component of a vector type, result[i] = if MSB of c[i] is set ? b[i] : a[i].

Each bit of the result is the corresponding bit of a if the corresponding bit of c is 0.

Returns 1 if the most significant bit in all components of x is set; otherwise returns 0.

Returns 1 if the most significant bit in any component of x is set; otherwise returns 0.

Test for sign bit.

Test if arguments are unordered.

Test if arguments are ordered.

Test for a normal value.

Test for a NaN.

Test for infinity value (+ve or –ve) .

Test for finite value.

Returns the component-wise compare of (x $<$ y) $||$ (x $>$ y) .

Returns the component-wise compare of x $<=$ y.

Returns the component-wise compare of x $<$ y.

Returns the component-wise compare of x $>=$ y.

Returns the component-wise compare of x $>$ y.

Returns the component-wise compare of x != y.

intn isequal (floatn x, floatn y) Returns the component-wise compare of x == y.

Returns the length of vector p computed as: half_sqrt(p.x2 + p.y2 + ...)

Returns fast_length(p0 – p1).

Returns a vector in the same direction as p but with a length of 1.

Return the length of vector p, i.e., sqrt(p.x2 + p.y 2 + ...)

Returns the distance between p0 and p1.

Compute dot product.

Returns the cross product of p0.xyz and p1.xyz.

Returns 1.0 if x $>$ 0, -0.0 if x = -0.0, +0.0 if x = +0.0, or –1.0 if x $<$ 0.

Returns 0.0 if x $<=$ edge0 and 1.0 if x $>=$ edge1 and performs smooth Hermite interpolation between 0 and 1when edge0 $<$ x $<$ edge1.

Returns 0.0 if x $<$ edge, otherwise it returns 1.0.

Converts degrees to radians, i.e.

Returns the linear blend of x & y implemented as: x + (y – x) $*$ a a must be a value in the range 0.0 . . . 1.0.

Converts radians to degrees, i.e.

Returns fmin(fmax(x, minval), maxval).

Multiply two 24-bit integer values x and y.

Multiply two 24-bit integer values x and y and add the 32-bit integer result to the 32-bit integer z.

result[i] = ((long)hi[i] $<<$ 32) | lo[i] result[i] = ((ulong)hi[i] $<<$ 32) | lo[i]

result[i] = ((int)hi[i] $<<$ 16) | lo[i] result[i] = ((uint)hi[i] $<<$ 16) | lo[i]

result[i] = ((short)hi[i] $<<$ 8) | lo[i] result[i] = ((ushort)hi[i] $<<$ 8) | lo[i]

Returns x - y and saturates the result.

For each element in v, the bits are shifted left by the number of bits given by the corresponding element in i (subject to usual shift modulo rules described in section 6.3).

Computes x $*$ y and returns the high half of the product of x and y.

Returns a $*$ b + c and saturates the result.

Returns mul_hi(a, b) + c.

Returns the number of leading 0-bits in x, starting at the most significant bit position.

Returns min(max(x, minval), maxval).

Returns (x + y + 1) $>>$ 1.

Returns (x + y) $>>$ 1.

Returns x + y and saturates the result.

Returns | x – y | without modulo overflow.

Returns | x |.

Compute tangent over an implementation-defined range.

Compute square root over an implementation-defined range.

Compute sine over an implementation-defined range.

Compute inverse square root over an implementationdefined range.

Compute reciprocal over an implementation-defined range.

Compute a base 10 logarithm over an implementationdefined range.

Compute a base 2 logarithm over an implementationdefined range.

Compute natural logarithm over an implementationdefined range.

Compute the base- 10 exponential of x over an implementation-defined range.

Compute the base- 2 exponential of x over an implementation-defined range.

Compute the base- e exponential of x over an implementation-defined range.

Compute x / y over an implementation-defined range.

Compute cosine over an implementation-defined range.

Compute reciprocal.

Compute the base- 10 exponential of x.

Compute the base- 2 exponential of x.

Compute x / y.

Round to integral value using the round to zero rounding mode.

Compute the gamma function.

Compute tan (PI $*$ x).

Compute hyperbolic tangent.

Compute tangent.

Compute square root.

Compute sin (PI $*$ x).

Compute hyperbolic sine.

Compute sine and cosine of x.

Compute sine.

Compute inverse square root.

Return the integral value nearest to x rounding halfway cases away from zero, regardless of the current rounding direction.

Compute x to the power 1/y.

Round to integral value (using round to nearest even rounding mode) in floating-point format.

The remquo function computes the value r such that r = x - n∗y, where n is the integer nearest the exact value of x/y.

Compute the value r such that r = x - n∗y, where n is the integer nearest the exact value of x/y.

Compute x to the power y, where x is >= 0.

Compute x to the power y, where y is an integer.

Compute x to the power y.

Computes the next representable single-precision floating-point value following x in the direction of y.

Returns a quiet NaN.

Decompose a floating-point number.

Returns x if | x | < | y |, y if | y | < | x |, otherwise fmin(x, y).

Returns x if | x | > | y |, y if | y | > | x |, otherwise fmax(x, y).

mad approximates a ∗ b + c.

Compute the exponent of x, which is the integral part of logr | x |.

Compute a base e logarithm of (1.0 + x).

Compute a base 10 logarithm.

Compute a base 2 logarithm.

Compute natural logarithm.

Log gamma function.

Multiply x by 2 to the power n.

Return the exponent as an integer value.

Compute the value of the square root of x^2+ y^2 without undue overflow or underflow.

Extract mantissa and exponent from x.

Returns fmin(x – floor (x), 0x1.fffffep-1f).

Modulus.

Returns y if y < x, otherwise it returns x.

Returns y if x < y, otherwise it returns x.

Returns the correctly rounded floating-point representation of the sum of c with the infinitely precise product of a and b.

Round to integral value using the round to –ve infinity rounding mode.

x - y if x > y, +0 if x is less than or equal to y.

Compute absolute value of a floating-point number.

Compute e^x- 1.0.

Exponential base 10 function.

Exponential base 2 function.

Compute the base- e exponential of x.

Error function encountered in integrating the normal distribution.

Complementary error function.

Compute cos (PI ∗ x).

Compute hyperbolic consine.

Compute cosine.

Returns x with its sign changed to match the sign of y.

Round to integral value using the round to +ve infinity rounding mode.

Compute cube-root.

Compute atan2 (y, x) / PI.

Compute atan (x) / PI.

Hyperbolic arc tangent.

Arc tangent of y / x.

Arc tangent function.

Compute asin (x) / PI.

Inverse hyperbolic sine.

Arc sine function.

Compute acos (x) / PI.

Inverse hyperbolic cosine.

Arc cosine function.

get_global_offset returns the offset values specified in global_work_offset argument to clEnqueueNDRangeKernel.

get_group_id returns the work-group ID which is a number from 0 .

Returns the number of work-groups that will execute a kernel for dimension identified by dimindx.

Returns the unique local work-item ID i.e.

Returns the number of local work-items specified in dimension identified by dimindx.

Returns the unique global work-item ID value for dimension identified by dimindx.

Returns the number of global work-items specified for dimension identified by dimindx.

This is the value given to the work_dim argument specified in clEnqueueNDRangeKernel. For clEnqueueTask, this returns 1.

This value is given by the global_work_size argument to clEnqueueNDRangeKernel. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_global_size() returns 1. For clEnqueueTask, this always returns 1.

The global work-item ID specifies the work-item ID based on the number of global work-items specified to execute the kernel. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_global_id() returns 0. For clEnqueueTask, this returns 0.

This value is given by the local_work_size argument to clEnqueueNDRangeKernel if local_work_size is not NU↩ LL; otherwise the OpenCL implementation chooses an appropriate local_work_size value which is returned by this function. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_local_size() returns 1. For clEnqueueTask, this always returns 1.

a work-item within a specific work-group for dimension identified by dimindx. Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_local_id() returns 0. For clEnqueueTask, this returns 0.

Valid values of dimindx are 0 to get_work_dim() – 1. For other values of dimindx, get_num_groups () returns 1. For clEnqueueTask, this always returns 1.

. get_num_groups(dimindx) – 1. Valid values of dimindx are 0 to get_work_dim() – 1. For other values, get_group↩ _id() returns 0. For clEnqueueTask, this returns 0.

Valid values of dimindx are 0 to get_work_dim() – 1. For other values, get_global_offset() returns 0. For cl↩ EnqueueTask, this returns 0.

Rounding of intermediate products shall not occur. Edge case behavior is per the IEEE 754-2008 standard.

If one argument is a NaN, fmax() returns the other argument. If both arguments are NaNs, fmax() returns a NaN.

If one argument is a NaN, fmin() returns the other argument. If both arguments are NaNs, fmin() returns a NaN.

Returns x – y ∗ trunc (x/y).

floor(x) is returned in iptr.

For each component the mantissa returned is a float with magnitude in the interval [1/2, 1) or 0. Each component of x equals mantissa returned ∗ 2^exp.

Returns the natural logarithm of the absolute value of the gamma function. The sign of the gamma function is returned in the signp argument of lgamma_r.

Whether or how the product of a ∗ b is rounded and how supernormal or subnormal intermediate products are handled is not defined. mad is intended to be used where speed is preferred over accuracy.

The modf function breaks the argument x into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part in the object pointed to by iptr.

The nancode may be placed in the significand of the resulting NaN.

Thus, if y is less than x, nextafter() returns the largest representable floating-point number less than x.

If there are two integers closest to x/y, n shall be the even one. If r is zero, it is given the same sign as x.

If there are two integers closest to x/y, n shall be the even one. If r is zero, it is given the same sign as x. This is the same value that is returned by the remainder function. remquo also calculates the lower seven bits of the integral quotient x/y, and gives that value the same sign as x/y. It stores this signed value in the object pointed to by quo.

Refer to section 7.1 for description of rounding modes.

The computed sine is the return value and computed cosine is returned in cosval.

x must be in the range -2^16 ... +2^16.

x must be in the range -2^16 . . . +2^16.

x must be in the range -216 . . . +216.

The maximum error is implementation-defined.

The maximum error is implementation defined.

The maximum error is implementationdefined.

The range of x and y are implementation-defined. The maximum error is implementation-defined.

The intermediate sum does not modulo overflow.

Results are undefined if minval > maxval.

Bits shifted off the left side of the element are shifted back in from the right.

Refer to definition of mul24 to see how the 24-bit integer multiplication is performed.

x and y are 32-bit integers but only the low 24-bits are used to perform the multiplication. mul24 should only be used when values in x and y are in the range [- 2^23, 2^23-1] if x and y are signed integers and in the range [0, 2^24-1] if x and y are unsigned integers. If x and y are not in this range, the multiplication result is implementation-defined.

(180 / PI) ∗ radians.

If x and y are infinite or NaN, the return values are undefined.

If a is not in the range 0.0 . . . 1.0, the return values are undefined.

(PI / 180) ∗ degrees.

This is useful in cases where you would want a threshold function with a smooth transition. This is equivalent to: gentype t; t = clamp ((x – edge0) / (edge1 – edge0), 0, 1); return t ∗ t ∗ (3 – 2 ∗ t); Results are undefined if edge0 >= edge1 or if x, edge0 or edge1 is a NaN.

Returns 0.0 if x is a NaN.

The w component of float4 result returned will be 0.0.

This is calculated as length(p0 – p1).

fast_normalize is computed as: p ∗ half_rsqrt (p.x^2 + p.y^2 + . . .) The result shall be within 8192 ulps error from the infinitely precise result of if (all(p == 0.0f)) result = p; else result = p / sqrt (p.x^2 + p.y^2 + ...); with the following exceptions: 1) If the sum of squares is greater than FLT_MAX then the value of the floating-point values in the result vector are undefined. 2) If the sum of squares is less than FLT_MIN then the implementation may return back p. 3) If the device is in “denorms are flushed to zero” mode, individual operand elements with magnitude less than sqrt(FLT_MIN) may be flushed to zero before proceeding with the calculation.

isordered() takes arguments x and y, and returns the result isequal(x, x) && isequal(y, y).

isunordered() takes arguments x and y, returning non-zero if x or y is NaN, and zero otherwise.

The scalar version of the function returns a 1 if the sign bit in the float is set else returns 0. The vector version of the function returns the following for each component in floatn: a -1 if the sign bit in the float is set else returns 0.

Otherwise it is the corresponding bit of b.

For a scalar type, result = c ? b : a.

The address computed as (p + (offset ∗ n)) must be 8-bit aligned if gentype is char, uchar; 16-bit aligned if gentype is short, ushort; 32-bit aligned if gentype is int, uint, float; 64-bit aligned if gentype is long, ulong.

The data read is interpreted as a half value. The half value is converted to a float value and the float value is returned. The read address computed as (p + offset) must be 16-bit aligned.

The data read is interpreted as a halfn value. The halfn value read is converted to a floatn value and the floatn value is returned. The read address computed as (p + (offset ∗ n)) must be 16-bit aligned.

The half value is then written to address computed as (p + offset). The address computed as (p + offset) must be 16- bit aligned. vstore_half use the current rounding mode. The default current rounding mode is round to nearest even.

The halfn value is then written to address computed as (p + (offset ∗ n)). The address computed as (p + (offset ∗ n)) must be 16-bit aligned. vstore_halfn uses the current rounding mode. The default current rounding mode is round to nearest even.

The data read is interpreted as a halfn value. The halfn value read is converted to a floatn value and the floatn value is returned. The address computed as (p + (offset ∗ n)) must be aligned to sizeof (halfn) bytes. For n = 3, vloada_half3 reads a half3 from address (p + (offset ∗ 4)) and returns a float3. The address computed as (p + (offset ∗ 4)) must be aligned to sizeof (half) ∗ 4 bytes.

For n = 1, 2, 4, 8 and 16, the halfn value is written to the address computed as (p + (offset

- n)). The address computed as (p + (offset ∗ n)) must be aligned to sizeof (halfn) bytes. For n = 3, the half3 value is written to the address computed as (p + (offset ∗ 4)). The address computed as (p + (offset ∗ 4)) must be aligned to sizeof (half) ∗ 4 bytes. vstorea_halfn uses the current rounding mode. The default current rounding mode is round to nearest even.

  This means that loads and stores preceding the mem_fence will be committed to memory before any loads and stores following the mem_fence. The flags argument specifies the memory address space and can be set to a combination of the following literal values: CLK_LOCAL_MEM_FENCE CLK_GLOBAL_MEM_FENCE.

The flags argument specifies the memory address space and can be set to to a combination of the following literal values: CLK_LOCAL_MEM_FENCE CLK_GLOBAL_MEM_FENCE.

The async copy is performed by all work-items in a workgroup and this built-in function must therefore be encountered by all work-items in a workgroup executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by wait_group_events to wait for the async copy to finish. The event argument can also be used to associate the async_work_group_copy with a previous async copy allowing an event to be shared by multiple async copies; otherwise event should be zero. If event argument is non-zero, the event object supplied in event argument will be returned. This function does not perform any implicit synchronization of source data such as using a barrier before performing the copy.

The src_stride is the stride in elements for each gentype element read from src. The dst_stride is the stride in elements for each gentype element written to dst. The async gather is performed by all work-items in a work-group. This built-in function must therefore be encountered by all work-items in a work-group executing the kernel with the same argument values; otherwise the results are undefined. Returns an event object that can be used by wait_group_events to wait for the async copy to finish. The event argument can also be used to associate the async_work_group_strided_copy with a previous async copy allowing an event to be shared by multiple async copies; otherwise event should be zero. If event argument is non-zero, the event object supplied in event argument

will be returned. This function does not perform any implicit synchronization of source data such as using a barrier before performing the copy.

The event objects specified in event_list will be released after the wait is performed. This function must be encountered by all workitems in a work-group executing the kernel with the same num_events and event objects specified in event_list; otherwise the results are undefined.

The prefetch instruction is applied to a work-item in a workgroup and does not affect the functional behavior of the kernel.

Compute (old + val) and store result at location pointed by p. The function returns old.

Compute (old - val) and store result at location pointed by p. The function returns old.

Returns old value.

Compute (old + 1) and store result at location pointed by p. The function returns old.

Compute (old - 1) and store result at location pointed by p. The function returns old.

Compute (old == cmp) ? val : old and store result at location pointed by p. The function returns old.

Compute min(old, val) and store minimum value at location pointed by p. The function returns old.

Compute max(old, val) and store maximum value at location pointed by p. The function returns old.

Compute (old & val) and store result at location pointed by p. The function returns old.

Compute (old | val) and store result at location pointed by p. The function returns old.

Compute (old ^ val) and store result at location pointed by p. The function returns old.

The size of each element in the mask must match the size of each element in the result. For shuffle, only the ilogb(2m-1) least significant bits of each mask element are considered. For shuffle2, only the ilogb(2m-1)+1 least significant bits of each mask element are considered. Other bits in the mask shall be ignored. The elements of the input vectors are numbered from left to right across one or both of the vectors. For this purpose, the number of elements in a vector is given by vec_step(gentypem). The shuffle mask operand specifies, for each element of the result vector, which element of the one or two input vectors the result element gets. Examples: uint4 mask = (uint4)(3, 2, 1, 0); float4 a; float4 r = shuffle(a, mask); // r.s0123 = a.wzyx uint8 mask = (uint8)(0, 1, 2, 3, 4, 5, 6, 7); float4 a, b; float8 r = shuffle2(a, b, mask); // r.s0123 = a.xyzw // r.s4567 = b.xyzw uint4 mask; float8 a; float4 b; b = shuffle(a, mask); Examples that are not valid are: uint8 mask; short16 a; short8 b; b = shuffle(a, mask); <- not valid

read_imagei and read_imageui return unnormalized signed integer and unsigned integer values respectively. Each channel will be stored in a 32-bit integer. read_imagei can only be used with image objects created with image↩_channel_data_type set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGN↩ED_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imagei are undefined. read_imageui can only be used with image objects created with image_channel_data_type set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNED_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imageui are undefined. The read_image{i|ui} calls support a nearest filter only. The filter_mode specified in sampler must be set to CLK_FI↩LTER_NEAREST; otherwise the values returned are undefined. Furthermore, the read_image{i|ui} calls that take integer coordinates must use a sampler with normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRES↩S_NONE; otherwise the values returned are undefined.

Appropriate data format conversion to the specified image format is done before writing the color value. x & y are considered to be unnormalized coordinates and must be in the range 0 ... image width - 1, and 0 ... image height - 1. write_imagef can only be used with image objects created with image_channel_data_type set to one of the

pre-defined packed formats or set to CL_SNORM_INT8, CL_UNORM_INT8, CL_SNORM_INT16, CL_UNORM_↩ INT16, CL_HALF_FLOAT or CL_FLOAT. Appropriate data format conversion will be done to convert channel data from a floating-point value to actual data format in which the channels are stored. write_imagei can only be used with image objects created with image_channel_data_type set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGNED_INT32. write_imageui can only be used with image objects created with image_channel_data_type set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNED_INT32. The behavior of write_imagef, write_imagei and write_imageui for image objects created with image_channel_data_type values not specified in the description above or with (x, y) coordinate values that are not in the range (0 ... image width - 1, 0 ... image height - 1), respectively, is undefined.

coord.w is ignored. read_imagef returns floating-point values in the range [0.0 ... 1.0] for image objects created with image_channel_data_type set to one of the predefined packed formats or CL_UNORM_INT8, or CL_UNORM_I↩ NT16. read_imagef returns floating-point values in the range [-1.0 ... 1.0] for image objects created with image↩ _channel_data_type set to CL_SNORM_INT8, or CL_SNORM_INT16. read_imagef returns floating-point values for image objects created with image_channel_data_type set to CL_HALF_FLOAT or CL_FLOAT. The read_↩ imagef calls that take integer coordinates must use a sampler with filter mode set to CLK_FILTER_NEAREST, normalized coordinates set to CLK_NORMALIZED_COORDS_FALSE and addressing mode set to CLK_ADDR↩ ESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined. Values returned by read_imagef for image objects with image_channel_data_type values not specified in the description are undefined.

coord.w is ignored. read_imagei and read_imageui return unnormalized signed integer and unsigned integer values respectively. Each channel will be stored in a 32-bit integer. read_imagei can only be used with image objects created with image_channel_data_type set to one of the following values: CL_SIGNED_INT8, CL_SIGNED_INT16 and CL_SIGNED_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imagei are undefined. read_imageui can only be used with image objects created with image_channel_↩ data_type set to one of the following values: CL_UNSIGNED_INT8, CL_UNSIGNED_INT16 and CL_UNSIGNE↩ D_INT32. If the image_channel_data_type is not one of the above values, the values returned by read_imageui are undefined. The read_image{i|ui} calls support a nearest filter only. The filter_mode specified in sampler must be set to CLK_FILTER_NEAREST; otherwise the values returned are undefined. Furthermore, the read_image{i|ui} calls that take integer coordinates must use a sampler with normalized coordinates set to CLK_NORMALIZED_↩ COORDS_FALSE and addressing mode set to CLK_ADDRESS_CLAMP_TO_EDGE, CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE; otherwise the values returned are undefined.

Valid values are: CLK_A CLK_R CLK_Rx CLK_RG CLK_RGx CLK_RA CLK_RGB CLK_RGBx CLK_RGBA CLK↩ _ARGB CLK_BGRA CLK_INTENSITY CLK_LUMINANCE

The width is returned in the x component, and the height in the y component.

The width is returned in the x component, height in the y component, depth in the z component and the w component is 0.

11.63.3.2 printf()

```
int printf (
            __constant const char * st,
             ...  )
```

11.63.4 Variable Documentation

11.63.4.1 a

```
double16 const_func double16 double a
```

Definition at line 3613 of file opencl_spir.h.

Referenced by cl::sycl::drt::accessor< Accessor >::accessor().

11.63.4.2 b

```
double16 const_func double16 b
```

Definition at line 915 of file opencl_spir.h.

Referenced by cl::sycl::detail::task::release_buffers().

11.63.4.3 c

```
double16 const_func double16 ulong16 c
```

Definition at line 915 of file opencl_spir.h.

11.63.4.4 cmp

```
unsigned int unsigned int cmp
```

Definition at line 6359 of file opencl_spir.h.

11.63.4.5 color

```
void int4 uint4 color
```

Definition at line 7138 of file opencl_spir.h.

11.63.4.6 coord

```
void int4 coord
```

Definition at line 7027 of file opencl_spir.h.

11.63.4.7 cosval

```
double16 __private double16 * cosval
```

Definition at line 1892 of file opencl_spir.h.

11.63.4.8 dst_stride

```
event_t const __local half16 size_t size_t dst_stride
```

Definition at line 6074 of file opencl_spir.h.

11.63.4.9 edge1

```
double16 const_func double16 edge1
```

Definition at line 3722 of file opencl_spir.h.

11.63.4.10 event

```
event_t const __local half16 size_t size_t event_t event
```

Definition at line 5863 of file opencl_spir.h.

11.63.4.11 event_list

```
void event_t* event_list
```

Definition at line 6162 of file opencl_spir.h.

11.63.4.12 exp

```
double16 __private int16 * exp
```

Definition at line 1102 of file opencl_spir.h.

Referenced by cl::sycl::TRISYCL_MATH_WRAP2s().

11.63.4.13 i

```
ulong16 const_func ulong16 i
```

Definition at line 3222 of file opencl_spir.h.

Referenced by cl::sycl::vec< DataType, NumElements >::flatten(), cl::sycl::vec< DataType, NumElements >↩ ::flatten_to_tuple(), cl::sycl::detail::cache< cl_device_id, cl::sycl::detail::opencl_device >::get_or_register(), cl↩ ::sycl::detail::linear_id(), cl::sycl::make_id(), cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >↩ ::operator[ ]() , cl::sycl::group< Dimensions >::parallel_for_work_item(), cl::sycl::detail::parallel_for_workitem(), and cl::sycl::detail::pipe< value_type >::reserve_write().

11.63.4.14 iptr

```
double16 __private double16 * iptr
```

Definition at line 1040 of file opencl_spir.h.

11.63.4.15 lo

```
ulong16 const_func uint16 lo
```

Definition at line 3327 of file opencl_spir.h.

11.63.4.16 mask

```
double16 const_func double16 ulong16 mask
```

Definition at line 6535 of file opencl_spir.h.

11.63.4.17 maxval

```
double16 const_func double double maxval
```

Definition at line 2707 of file opencl_spir.h.

11.63.4.18 minval

```
double16 const_func double minval
```

Definition at line 2707 of file opencl_spir.h.

11.63.4.19 n

```
double16 const_func int n
```

Definition at line 1205 of file opencl_spir.h.

11.63.4.20 num_elements

```
void size_t num_elements
```

Definition at line 5863 of file opencl_spir.h.

11.63.4.21 offset

```
void size_t offset
```

Definition at line 5062 of file opencl_spir.h.

Referenced by cl::sycl::detail::linear_id(), and cl::sycl::detail::parallel_for_global_offset().

11.63.4.22 p

```
void size_t __private half * p
```

Definition at line 4827 of file opencl_spir.h.

Referenced by cl::sycl::detail::address_space_ptr< T, AS >::address_space_ptr(), cl::sycl::detail::pipe_accessor< DataType, AccessMode, access::target::blocking_pipe >::pipe_accessor(), cl::sycl::detail::task::postlude(), cl::sycl::detail::task::prelude(), and cl::sycl::drt::code::program::program().

11.63.4.23 p1

```
float const_func float4 p1
```

Definition at line 3787 of file opencl_spir.h.

11.63.4.24 quo

```
double16 double16 __private int16 * quo
```

Definition at line 1717 of file opencl_spir.h.

11.63.4.25 sampler

```
uint4 const_func sampler_t sampler
```

Definition at line 7027 of file opencl_spir.h.

11.63.4.26 signp

```
double16 __private int16 * signp
```

Definition at line 1265 of file opencl_spir.h.

11.63.4.27 src

```
event_t const __local half16 * src
```

Definition at line 5863 of file opencl_spir.h.

Referenced by cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >::partition_array(), and cl::sycl↩ ::detail::small_array< BasicType, FinalType, 3 >::small_array().

11.63.4.28 src_stride

```
event_t const __global half16 size_t size_t src_stride
```

Definition at line 6020 of file opencl_spir.h.

11.63.4.29 val

```
unsigned int unsigned int val
```

Definition at line 6245 of file opencl_spir.h.

11.63.4.30 x

```
double16 const_func double16 double16 x
```

Definition at line 489 of file opencl_spir.h.

11.63.4.31 y

```
double16 const_func double16 y
```

Definition at line 623 of file opencl_spir.h.

11.63.4.32 z

```
uint16 const_func uint16 uint16 z
```

Definition at line 3431 of file opencl_spir.h.

11.64 opencl_spir.h

```
00001 //
00002 //                     SPIR Tools
00003 //
00004 // This file is distributed under the University of Illinois Open Source
00005 // License. See LICENSE.TXT for details.
00006 //
00007
00008 #ifndef _OPENCL_H_
00009 #define _OPENCL_H_
00010
00011 #if !defined(__SPIR32__) && !defined(__SPIR64__)
00012 #error "This header file should be used with SPIR target only."
00013 #endif
00014
00015 /// Added according to pocl/examples/example1-spir64/generate_spir.sh
00016 /// \todo avoid this hack
00017 #pragma OPENCL EXTENSION cl_khr_fp64 : enable
00018
00019 // Optimizations
00020
00021 #define const_func __attribute__((const))
00022 #define readonly __attribute__((pure))
00023
00024 // built-in scalar data types:
00025
00026 /**
00027  * An unsigned 8-bit integer.
```

```
 */
typedef unsigned char uchar;

/**
 * An unsigned 16-bit integer.
 */
typedef unsigned short ushort;

/**
 * An unsigned 32-bit integer.
 */
typedef unsigned int uint;

/**
 * An unsigned 64-bit integer.
 */
typedef unsigned long ulong;

/**
 * The unsigned integer type of the result of the sizeof operator. This
 * is a 32-bit unsigned integer if CL_DEVICE_ADDRESS_BITS
 * defined in table 4.3 is 32-bits and is a 64-bit unsigned integer if
 * CL_DEVICE_ADDRESS_BITS is 64-bits.
 */
#if defined(__SPIR32__)
typedef uint size_t;
#elif defined (__SPIR64__)
typedef ulong size_t;
#endif

/**
 * A signed integer type that is the result of subtracting two pointers.
 * This is a 32-bit signed integer if CL_DEVICE_ADDRESS_BITS
 * defined in table 4.3 is 32-bits and is a 64-bit signed integer if
 * CL_DEVICE_ADDRESS_BITS is 64-bits.
 */
#if defined(__SPIR32__)
typedef int ptrdiff_t;
#elif defined (__SPIR64__ )
typedef long ptrdiff_t;
#endif

typedef ptrdiff_t intptr_t;

typedef size_t uintptr_t;


// built-in vector data types:

typedef char char2 __attribute__((ext_vector_type(2)));
typedef char char3 __attribute__((ext_vector_type(3)));
typedef char char4 __attribute__((ext_vector_type(4)));
typedef char char8 __attribute__((ext_vector_type(8)));
typedef char char16 __attribute__((ext_vector_type(16)));
typedef uchar uchar2 __attribute__((ext_vector_type(2)));
typedef uchar uchar3 __attribute__((ext_vector_type(3)));
typedef uchar uchar4 __attribute__((ext_vector_type(4)));
typedef uchar uchar8 __attribute__((ext_vector_type(8)));
typedef uchar uchar16 __attribute__((ext_vector_type(16)));
typedef short short2 __attribute__((ext_vector_type(2)));
typedef short short3 __attribute__((ext_vector_type(3)));
typedef short short4 __attribute__((ext_vector_type(4)));
typedef short short8 __attribute__((ext_vector_type(8)));
typedef short short16 __attribute__((ext_vector_type(16)));
typedef ushort ushort2 __attribute__((ext_vector_type(2)));
typedef ushort ushort3 __attribute__((ext_vector_type(3)));
typedef ushort ushort4 __attribute__((ext_vector_type(4)));
typedef ushort ushort8 __attribute__((ext_vector_type(8)));
typedef ushort ushort16 __attribute__((ext_vector_type(16)));
typedef int int2 __attribute__((ext_vector_type(2)));
typedef int int3 __attribute__((ext_vector_type(3)));
typedef int int4 __attribute__((ext_vector_type(4)));
typedef int int8 __attribute__((ext_vector_type(8)));
typedef int int16 __attribute__((ext_vector_type(16)));
typedef uint uint2 __attribute__((ext_vector_type(2)));
typedef uint uint3 __attribute__((ext_vector_type(3)));
typedef uint uint4 __attribute__((ext_vector_type(4)));
typedef uint uint8 __attribute__((ext_vector_type(8)));
typedef uint uint16 __attribute__((ext_vector_type(16)));
typedef long long2 __attribute__((ext_vector_type(2)));
typedef long long3 __attribute__((ext_vector_type(3)));
typedef long long4 __attribute__((ext_vector_type(4)));
typedef long long8 __attribute__((ext_vector_type(8)));
typedef long long16 __attribute__((ext_vector_type(16)));
typedef ulong ulong2 __attribute__((ext_vector_type(2)));
typedef ulong ulong3 __attribute__((ext_vector_type(3)));
typedef ulong ulong4 __attribute__((ext_vector_type(4)));
```

```
typedef ulong ulong8 __attribute__((ext_vector_type(8)));
typedef ulong ulong16 __attribute__((ext_vector_type(16)));
typedef float float2 __attribute__((ext_vector_type(2)));
typedef float float3 __attribute__((ext_vector_type(3)));
typedef float float4 __attribute__((ext_vector_type(4)));
typedef float float8 __attribute__((ext_vector_type(8)));
typedef float float16 __attribute__((ext_vector_type(16)));
typedef half half2 __attribute__((ext_vector_type(2)));
typedef half half3 __attribute__((ext_vector_type(3)));
typedef half half4 __attribute__((ext_vector_type(4)));
typedef half half8 __attribute__((ext_vector_type(8)));
typedef half half16 __attribute__((ext_vector_type(16)));
typedef double double2 __attribute__((ext_vector_type(2)));
typedef double double3 __attribute__((ext_vector_type(3)));
typedef double double4 __attribute__((ext_vector_type(4)));
typedef double double8 __attribute__((ext_vector_type(8)));
typedef double double16 __attribute__((ext_vector_type(16)));

/**
 * Value of maximum non-infinite single-precision floating-point
 * number.
 */
#define MAXFLOAT    0x1.fffffep127f

/**
 * A positive float constant expression. HUGE_VALF evaluates
 * to +infinity. Used as an error value returned by the built-in
 * math functions.
 */
#define HUGE_VALF (__builtin_huge_valf())

/**
 * A positive double constant expression. HUGE_VAL evaluates
 * to +infinity. Used as an error value returned by the built-in
 * math functions.
 */
#define HUGE_VAL (__builtin_huge_val())

/**
 * A constant expression of type float representing positive or
 * unsigned infinity.
 */
#define INFINITY (__builtin_inff())

/**
 * A constant expression of type float representing a quiet NaN.
 */
#define NAN as_float(__builtin_nanf((char const *)""))

#define FP_ILOGB0        INT_MIN
#define FP_ILOGBNAN      INT_MAX

#define FLT_DIG 6
#define FLT_MANT_DIG 24
#define FLT_MAX_10_EXP +38
#define FLT_MAX_EXP +128
#define FLT_MIN_10_EXP -37
#define FLT_MIN_EXP -125
#define FLT_RADIX 2
#define FLT_MAX 0x1.fffffep127f
#define FLT_MIN 0x1.0p-126f
#define FLT_EPSILON 0x1.0p-23f

#define DBL_DIG 15
#define DBL_MANT_DIG 53
#define DBL_MAX_10_EXP +308
#define DBL_MAX_EXP +1024
#define DBL_MIN_10_EXP -307
#define DBL_MIN_EXP -1021
#define DBL_RADIX 2
#define DBL_MAX 0x1.fffffffffffffp1023
#define DBL_MIN 0x1.0p-1022
#define DBL_EPSILON 0x1.0p-52


#define M_E_F         2.71828182845904523536028747135266250f
#define M_LOG2E_F     1.44269504088896340735992468100189214f
#define M_LOG10E_F    0.43429448190325182765112891891660508 2f
#define M_LN2_F       0.69314718055994530941723212145817656 8f
#define M_LN10_F      2.30258509299404568401799145468436421f
#define M_PI_F        3.14159265358979323846264338327950288f
#define M_PI_2_F      1.57079632679489661923132169163975144f
#define M_PI_4_F      0.78539816339744830961566084581987572 1f
#define M_1_PI_F      0.31830988618379067153776752674502872 4f
#define M_2_PI_F      0.63661977236758134307553505349005744 8f
#define M_2_SQRTPI_F  1.12837916709551257389615890312154517f
#define M_SQRT2_F     1.41421356237309504880168872420969808f
```

```
#define M_SQRT1_2_F   0.707106781186547524400844362104849039f

#define M_E             0x1.5bf0a8b145769p+1
#define M_LOG2E         0x1.71547652b82fep+0
#define M_LOG10E        0x1.bcb7b1526e50ep-2
#define M_LN2           0x1.62e42fefa39efp-1
#define M_LN10          0x1.26bb1bbb55516p+1
#define M_PI            0x1.921fb54442d18p+1
#define M_PI_2          0x1.921fb54442d18p+0
#define M_PI_4          0x1.921fb54442d18p-1
#define M_1_PI          0x1.45f306dc9c883p-2
#define M_2_PI          0x1.45f306dc9c883p-1
#define M_2_SQRTPI      0x1.20dd750429b6dp+0
#define M_SQRT2         0x1.6a09e667f3bcdp+0
#define M_SQRT1_2       0x1.6a09e667f3bcdp-1

#define CHAR_BIT    8
#define SCHAR_MAX   127
#define SCHAR_MIN   (-128)
#define UCHAR_MAX   255
#define CHAR_MAX    SCHAR_MAX
#define CHAR_MIN    SCHAR_MIN
#define USHRT_MAX   65535
#define SHRT_MAX    32767
#define SHRT_MIN    (-32768)
#define UINT_MAX    0xffffffff
#define INT_MAX     2147483647
#define INT_MIN     (-2147483647-1)
#define ULONG_MAX   0xffffffffffffffffUL
#define LONG_MAX    ((long)0x7fffffffffffffffL)
#define LONG_MIN    ((long)(-0x7fffffffffffffffL-1))

#define DBL_DIG 15
#define DBL_MANT_DIG 53
#define DBL_MAX_10_EXP +308
#define DBL_MAX_EXP +1024
#define DBL_MIN_10_EXP -307
#define DBL_MIN_EXP -1021
#define DBL_MAX 0x1.fffffffffffffp1023
#define DBL_MIN 0x1.0p-1022
#define DBL_EPSILON 0x1.0p-52

// work-item functions

/**
 * Returns the number of dimensions in use. This is the
 * value given to the work_dim argument specified in
 * clEnqueueNDRangeKernel.
 * For clEnqueueTask, this returns 1.
 */
uint const_func __attribute__((overloadable)) get_work_dim(void);

/**
 * Returns the number of global work-items specified for
 * dimension identified by dimindx. This value is given by
 * the global_work_size argument to
 * clEnqueueNDRangeKernel. Valid values of dimindx
 * are 0 to get_work_dim() - 1. For other values of
 * dimindx, get_global_size() returns 1.
 * For clEnqueueTask, this always returns 1.
 */
size_t const_func __attribute__((overloadable)) get_global_size(uint dimindx);

/**
 * Returns the unique global work-item ID value for
 * dimension identified by dimindx. The global work-item
 * ID specifies the work-item ID based on the number of
 * global work-items specified to execute the kernel. Valid
 * values of dimindx are 0 to get_work_dim() - 1. For
 * other values of dimindx, get_global_id() returns 0.
 * For clEnqueueTask, this returns 0.
 */
size_t const_func __attribute__((overloadable)) get_global_id(uint dimindx);

/**
 * Returns the number of local work-items specified in
 * dimension identified by dimindx. This value is given by
 * the local_work_size argument to
 * clEnqueueNDRangeKernel if local_work_size is not
 * NULL; otherwise the OpenCL implementation chooses
 * an appropriate local_work_size value which is returned
 * by this function. Valid values of dimindx are 0 to
 * get_work_dim() - 1. For other values of dimindx,
 * get_local_size() returns 1.
 * For clEnqueueTask, this always returns 1.
```

```
 */
size_t const_func __attribute__((overloadable)) get_local_size(uint dimindx);

/**
 * Returns the unique local work-item ID i.e. a work-item
 * within a specific work-group for dimension identified by
 * dimindx. Valid values of dimindx are 0 to
 * get_work_dim() - 1. For other values of dimindx,
 * get_local_id() returns 0.
 * For clEnqueueTask, this returns 0.
 */
size_t const_func __attribute__((overloadable)) get_local_id(uint dimindx);

/**
 * Returns the number of work-groups that will execute a
 * kernel for dimension identified by dimindx.
 * Valid values of dimindx are 0 to get_work_dim() - 1.
 * For other values of dimindx, get_num_groups () returns
 * 1.
 * For clEnqueueTask, this always returns 1.
 */
size_t const_func __attribute__((overloadable)) get_num_groups(uint dimindx);

/**
 * get_group_id returns the work-group ID which is a
 * number from 0 .. get_num_groups(dimindx) - 1.
 * Valid values of dimindx are 0 to get_work_dim() - 1.
 * For other values, get_group_id() returns 0.
 * For clEnqueueTask, this returns 0.
 */
size_t const_func __attribute__((overloadable)) get_group_id(uint dimindx);

/**
 * get_global_offset returns the offset values specified in
 * global_work_offset argument to
 * clEnqueueNDRangeKernel.
 * Valid values of dimindx are 0 to get_work_dim() - 1.
 * For other values, get_global_offset() returns 0.
 * For clEnqueueTask, this returns 0.
 */
size_t const_func __attribute__((overloadable)) get_global_offset(uint dimindx);

int printf(__constant const char* st, ...);
// Math functions:

/**
 * Arc cosine function.
 */
float const_func __attribute__((overloadable)) acos(float);
float2 const_func __attribute__((overloadable)) acos(float2);
float3 const_func __attribute__((overloadable)) acos(float3);
float4 const_func __attribute__((overloadable)) acos(float4);
float8 const_func __attribute__((overloadable)) acos(float8);
float16 const_func __attribute__((overloadable)) acos(float16);
double const_func __attribute__((overloadable)) acos(double);
double2 const_func __attribute__((overloadable)) acos(double2);
double3 const_func __attribute__((overloadable)) acos(double3);
double4 const_func __attribute__((overloadable)) acos(double4);
double8 const_func __attribute__((overloadable)) acos(double8);
double16 const_func __attribute__((overloadable)) acos(double16);
//half const_func __attribute__((overloadable)) acos(half);
//half2 const_func __attribute__((overloadable)) acos(half2);
//half3 const_func __attribute__((overloadable)) acos(half3);
//half4 const_func __attribute__((overloadable)) acos(half4);
//half8 const_func __attribute__((overloadable)) acos(half8);
//half16 const_func __attribute__((overloadable)) acos(half16);

/**
 * Inverse hyperbolic cosine.
 */
float const_func __attribute__((overloadable)) acosh(float);
float2 const_func __attribute__((overloadable)) acosh(float2);
float3 const_func __attribute__((overloadable)) acosh(float3);
float4 const_func __attribute__((overloadable)) acosh(float4);
float8 const_func __attribute__((overloadable)) acosh(float8);
float16 const_func __attribute__((overloadable)) acosh(float16);
double const_func __attribute__((overloadable)) acosh(double);
double2 const_func __attribute__((overloadable)) acosh(double2);
double3 const_func __attribute__((overloadable)) acosh(double3);
double4 const_func __attribute__((overloadable)) acosh(double4);
double8 const_func __attribute__((overloadable)) acosh(double8);
double16 const_func __attribute__((overloadable)) acosh(double16);
```

```
//half const_func __attribute__((overloadable)) acosh(half);
//half2 const_func __attribute__((overloadable)) acosh(half2);
//half3 const_func __attribute__((overloadable)) acosh(half3);
//half4 const_func __attribute__((overloadable)) acosh(half4);
//half8 const_func __attribute__((overloadable)) acosh(half8);
//half16 const_func __attribute__((overloadable)) acosh(half16);

/**
 * Compute acos (x) / PI.
 */
float const_func __attribute__((overloadable)) acospi(float x);
float2 const_func __attribute__((overloadable)) acospi(float2 x);
float3 const_func __attribute__((overloadable)) acospi(float3 x);
float4 const_func __attribute__((overloadable)) acospi(float4 x);
float8 const_func __attribute__((overloadable)) acospi(float8 x);
float16 const_func __attribute__((overloadable)) acospi(float16 x);
double const_func __attribute__((overloadable)) acospi(double x);
double2 const_func __attribute__((overloadable)) acospi(double2 x);
double3 const_func __attribute__((overloadable)) acospi(double3 x);
double4 const_func __attribute__((overloadable)) acospi(double4 x);
double8 const_func __attribute__((overloadable)) acospi(double8 x);
double16 const_func __attribute__((overloadable)) acospi(double16 x);
//half const_func __attribute__((overloadable)) acospi(half x);
//half2 const_func __attribute__((overloadable)) acospi(half2 x);
//half3 const_func __attribute__((overloadable)) acospi(half3 x);
//half4 const_func __attribute__((overloadable)) acospi(half4 x);
//half8 const_func __attribute__((overloadable)) acospi(half8 x);
//half16 const_func __attribute__((overloadable)) acospi(half16 x);

/**
 * Arc sine function.
 */
float const_func __attribute__((overloadable)) asin(float);
float2 const_func __attribute__((overloadable)) asin(float2);
float3 const_func __attribute__((overloadable)) asin(float3);
float4 const_func __attribute__((overloadable)) asin(float4);
float8 const_func __attribute__((overloadable)) asin(float8);
float16 const_func __attribute__((overloadable)) asin(float16);
double const_func __attribute__((overloadable)) asin(double);
double2 const_func __attribute__((overloadable)) asin(double2);
double3 const_func __attribute__((overloadable)) asin(double3);
double4 const_func __attribute__((overloadable)) asin(double4);
double8 const_func __attribute__((overloadable)) asin(double8);
double16 const_func __attribute__((overloadable)) asin(double16);
//half const_func __attribute__((overloadable)) asin(half);
//half2 const_func __attribute__((overloadable)) asin(half2);
//half3 const_func __attribute__((overloadable)) asin(half3);
//half4 const_func __attribute__((overloadable)) asin(half4);
//half8 const_func __attribute__((overloadable)) asin(half8);
//half16 const_func __attribute__((overloadable)) asin(half16);

/**
 * Inverse hyperbolic sine.
 */
float const_func __attribute__((overloadable)) asinh(float);
float2 const_func __attribute__((overloadable)) asinh(float2);
float3 const_func __attribute__((overloadable)) asinh(float3);
float4 const_func __attribute__((overloadable)) asinh(float4);
float8 const_func __attribute__((overloadable)) asinh(float8);
float16 const_func __attribute__((overloadable)) asinh(float16);
double const_func __attribute__((overloadable)) asinh(double);
double2 const_func __attribute__((overloadable)) asinh(double2);
double3 const_func __attribute__((overloadable)) asinh(double3);
double4 const_func __attribute__((overloadable)) asinh(double4);
double8 const_func __attribute__((overloadable)) asinh(double8);
double16 const_func __attribute__((overloadable)) asinh(double16);
//half const_func __attribute__((overloadable)) asinh(half);
//half2 const_func __attribute__((overloadable)) asinh(half2);
//half3 const_func __attribute__((overloadable)) asinh(half3);
//half4 const_func __attribute__((overloadable)) asinh(half4);
//half8 const_func __attribute__((overloadable)) asinh(half8);
//half16 const_func __attribute__((overloadable)) asinh(half16);

/**
 * Compute asin (x) / PI.
```

```
 */
float const_func __attribute__((overloadable)) asinpi(float x);
float2 const_func __attribute__((overloadable)) asinpi(float2 x);
float3 const_func __attribute__((overloadable)) asinpi(float3 x);
float4 const_func __attribute__((overloadable)) asinpi(float4 x);
float8 const_func __attribute__((overloadable)) asinpi(float8 x);
float16 const_func __attribute__((overloadable)) asinpi(float16 x);
double const_func __attribute__((overloadable)) asinpi(double x);
double2 const_func __attribute__((overloadable)) asinpi(double2 x);
double3 const_func __attribute__((overloadable)) asinpi(double3 x);
double4 const_func __attribute__((overloadable)) asinpi(double4 x);
double8 const_func __attribute__((overloadable)) asinpi(double8 x);
double16 const_func __attribute__((overloadable)) asinpi(double16 x);
//half const_func __attribute__((overloadable)) asinpi(half x);
//half2 const_func __attribute__((overloadable)) asinpi(half2 x);
//half3 const_func __attribute__((overloadable)) asinpi(half3 x);
//half4 const_func __attribute__((overloadable)) asinpi(half4 x);
//half8 const_func __attribute__((overloadable)) asinpi(half8 x);
//half16 const_func __attribute__((overloadable)) asinpi(half16 x);

/**
 * Arc tangent function.
 */
float const_func __attribute__((overloadable)) atan(float y_over_x);
float2 const_func __attribute__((overloadable)) atan(float2 y_over_x);
float3 const_func __attribute__((overloadable)) atan(float3 y_over_x);
float4 const_func __attribute__((overloadable)) atan(float4 y_over_x);
float8 const_func __attribute__((overloadable)) atan(float8 y_over_x);
float16 const_func __attribute__((overloadable)) atan(float16 y_over_x);
double const_func __attribute__((overloadable)) atan(double y_over_x);
double2 const_func __attribute__((overloadable)) atan(double2 y_over_x);
double3 const_func __attribute__((overloadable)) atan(double3 y_over_x);
double4 const_func __attribute__((overloadable)) atan(double4 y_over_x);
double8 const_func __attribute__((overloadable)) atan(double8 y_over_x);
double16 const_func __attribute__((overloadable)) atan(double16 y_over_x);
//half const_func __attribute__((overloadable)) atan(half y_over_x);
//half2 const_func __attribute__((overloadable)) atan(half2 y_over_x);
//half3 const_func __attribute__((overloadable)) atan(half3 y_over_x);
//half4 const_func __attribute__((overloadable)) atan(half4 y_over_x);
//half8 const_func __attribute__((overloadable)) atan(half8 y_over_x);
//half16 const_func __attribute__((overloadable)) atan(half16 y_over_x);

/**
 * Arc tangent of y / x.
 */
float const_func __attribute__((overloadable)) atan2(float y, float x);
float2 const_func __attribute__((overloadable)) atan2(float2 y, float2 x);
float3 const_func __attribute__((overloadable)) atan2(float3 y, float3 x);
float4 const_func __attribute__((overloadable)) atan2(float4 y, float4 x);
float8 const_func __attribute__((overloadable)) atan2(float8 y, float8 x);
float16 const_func __attribute__((overloadable)) atan2(float16 y, float16 x);
double const_func __attribute__((overloadable)) atan2(double y, double x);
double2 const_func __attribute__((overloadable)) atan2(double2 y, double2 x);
double3 const_func __attribute__((overloadable)) atan2(double3 y, double3 x);
double4 const_func __attribute__((overloadable)) atan2(double4 y, double4 x);
double8 const_func __attribute__((overloadable)) atan2(double8 y, double8 x);
double16 const_func __attribute__((overloadable)) atan2(double16 y, double16 x);
//half const_func __attribute__((overloadable)) atan2(half y, half x);
//half2 const_func __attribute__((overloadable)) atan2(half2 y, half2 x);
//half3 const_func __attribute__((overloadable)) atan2(half3 y, half3 x);
//half4 const_func __attribute__((overloadable)) atan2(half4 y, half4 x);
//half8 const_func __attribute__((overloadable)) atan2(half8 y, half8 x);
//half16 const_func __attribute__((overloadable)) atan2(half16 y, half16 x);
```

```

/**
 * Hyperbolic arc tangent.
 */
float const_func __attribute__((overloadable)) atanh(float);
float2 const_func __attribute__((overloadable)) atanh(float2);
float3 const_func __attribute__((overloadable)) atanh(float3);
float4 const_func __attribute__((overloadable)) atanh(float4);
float8 const_func __attribute__((overloadable)) atanh(float8);
float16 const_func __attribute__((overloadable)) atanh(float16);
double const_func __attribute__((overloadable)) atanh(double);
double2 const_func __attribute__((overloadable)) atanh(double2);
double3 const_func __attribute__((overloadable)) atanh(double3);
double4 const_func __attribute__((overloadable)) atanh(double4);
double8 const_func __attribute__((overloadable)) atanh(double8);
double16 const_func __attribute__((overloadable)) atanh(double16);
//half const_func __attribute__((overloadable)) atanh(half);
//half2 const_func __attribute__((overloadable)) atanh(half2);
//half3 const_func __attribute__((overloadable)) atanh(half3);
//half4 const_func __attribute__((overloadable)) atanh(half4);
//half8 const_func __attribute__((overloadable)) atanh(half8);
//half16 const_func __attribute__((overloadable)) atanh(half16);

/**
 * Compute atan (x) / PI.
 */
float const_func __attribute__((overloadable)) atanpi(float x);
float2 const_func __attribute__((overloadable)) atanpi(float2 x);
float3 const_func __attribute__((overloadable)) atanpi(float3 x);
float4 const_func __attribute__((overloadable)) atanpi(float4 x);
float8 const_func __attribute__((overloadable)) atanpi(float8 x);
float16 const_func __attribute__((overloadable)) atanpi(float16 x);
double const_func __attribute__((overloadable)) atanpi(double x);
double2 const_func __attribute__((overloadable)) atanpi(double2 x);
double3 const_func __attribute__((overloadable)) atanpi(double3 x);
double4 const_func __attribute__((overloadable)) atanpi(double4 x);
double8 const_func __attribute__((overloadable)) atanpi(double8 x);
double16 const_func __attribute__((overloadable)) atanpi(double16 x);
//half const_func __attribute__((overloadable)) atanpi(half x);
//half2 const_func __attribute__((overloadable)) atanpi(half2 x);
//half3 const_func __attribute__((overloadable)) atanpi(half3 x);
//half4 const_func __attribute__((overloadable)) atanpi(half4 x);
//half8 const_func __attribute__((overloadable)) atanpi(half8 x);
//half16 const_func __attribute__((overloadable)) atanpi(half16 x);

/**
 * Compute atan2 (y, x) / PI.
 */
float const_func __attribute__((overloadable)) atan2pi(float y, float x);
float2 const_func __attribute__((overloadable)) atan2pi(float2 y, float2 x);
float3 const_func __attribute__((overloadable)) atan2pi(float3 y, float3 x);
float4 const_func __attribute__((overloadable)) atan2pi(float4 y, float4 x);
float8 const_func __attribute__((overloadable)) atan2pi(float8 y, float8 x);
float16 const_func __attribute__((overloadable)) atan2pi(float16 y, float16 x);
double const_func __attribute__((overloadable)) atan2pi(double y, double x);
double2 const_func __attribute__((overloadable)) atan2pi(double2 y, double2 x);
double3 const_func __attribute__((overloadable)) atan2pi(double3 y, double3 x);
double4 const_func __attribute__((overloadable)) atan2pi(double4 y, double4 x);
double8 const_func __attribute__((overloadable)) atan2pi(double8 y, double8 x);
double16 const_func __attribute__((overloadable)) atan2pi(double16 y, double16 x);
//half const_func __attribute__((overloadable)) atan2pi(half y, half x);
//half2 const_func __attribute__((overloadable)) atan2pi(half2 y, half2 x);
//half3 const_func __attribute__((overloadable)) atan2pi(half3 y, half3 x);
```

```
//half4 const_func __attribute__((overloadable)) atan2pi(half4 y, half4 x);
//half8 const_func __attribute__((overloadable)) atan2pi(half8 y, half8 x);
//half16 const_func __attribute__((overloadable)) atan2pi(half16 y, half16 x);

/**
 * Compute cube-root.
 */
float const_func __attribute__((overloadable)) cbrt(float);
float2 const_func __attribute__((overloadable)) cbrt(float2);
float3 const_func __attribute__((overloadable)) cbrt(float3);
float4 const_func __attribute__((overloadable)) cbrt(float4);
float8 const_func __attribute__((overloadable)) cbrt(float8);
float16 const_func __attribute__((overloadable)) cbrt(float16);
double const_func __attribute__((overloadable)) cbrt(double);
double2 const_func __attribute__((overloadable)) cbrt(double2);
double3 const_func __attribute__((overloadable)) cbrt(double3);
double4 const_func __attribute__((overloadable)) cbrt(double4);
double8 const_func __attribute__((overloadable)) cbrt(double8);
double16 const_func __attribute__((overloadable)) cbrt(double16);
//half const_func __attribute__((overloadable)) cbrt(half);
//half2 const_func __attribute__((overloadable)) cbrt(half2);
//half3 const_func __attribute__((overloadable)) cbrt(half3);
//half4 const_func __attribute__((overloadable)) cbrt(half4);
//half8 const_func __attribute__((overloadable)) cbrt(half8);
//half16 const_func __attribute__((overloadable)) cbrt(half16);

/**
 * Round to integral value using the round to +ve
 * infinity rounding mode.
 */
float const_func __attribute__((overloadable)) ceil(float);
float2 const_func __attribute__((overloadable)) ceil(float2);
float3 const_func __attribute__((overloadable)) ceil(float3);
float4 const_func __attribute__((overloadable)) ceil(float4);
float8 const_func __attribute__((overloadable)) ceil(float8);
float16 const_func __attribute__((overloadable)) ceil(float16);
double const_func __attribute__((overloadable)) ceil(double);
double2 const_func __attribute__((overloadable)) ceil(double2);
double3 const_func __attribute__((overloadable)) ceil(double3);
double4 const_func __attribute__((overloadable)) ceil(double4);
double8 const_func __attribute__((overloadable)) ceil(double8);
double16 const_func __attribute__((overloadable)) ceil(double16);
//half const_func __attribute__((overloadable)) ceil(half);
//half2 const_func __attribute__((overloadable)) ceil(half2);
//half3 const_func __attribute__((overloadable)) ceil(half3);
//half4 const_func __attribute__((overloadable)) ceil(half4);
//half8 const_func __attribute__((overloadable)) ceil(half8);
//half16 const_func __attribute__((overloadable)) ceil(half16);

/**
 * Returns x with its sign changed to match the sign of
 * y.
 */
float const_func __attribute__((overloadable)) copysign(float
x, float y);
float2 const_func __attribute__((overloadable)) copysign(float2
x, float2 y);
float3 const_func __attribute__((overloadable)) copysign(float3
x, float3 y);
float4 const_func __attribute__((overloadable)) copysign(float4
x, float4 y);
float8 const_func __attribute__((overloadable)) copysign(float8
x, float8 y);
float16 const_func __attribute__((overloadable)) copysign(float16
x, float16 y);
double const_func __attribute__((overloadable)) copysign(double
x, double y);
double2 const_func __attribute__((overloadable)) copysign(double2
x, double2 y);
double3 const_func __attribute__((overloadable)) copysign(double3
x, double3 y);
double4 const_func __attribute__((overloadable)) copysign(double4
x, double4 y);
double8 const_func __attribute__((overloadable)) copysign(double8
x, double8 y);
double16 const_func __attribute__((overloadable)) copysign(double16
x, double16 y);
//half const_func __attribute__((overloadable)) copysign(half x, half y);
//half2 const_func __attribute__((overloadable)) copysign(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) copysign(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) copysign(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) copysign(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) copysign(half16 x, half16 y);

/**
 * Compute cosine.
 */
```

```
double const_func __attribute__((overloadable)) cos(double);
double2 const_func __attribute__((overloadable)) cos(double2);
double3 const_func __attribute__((overloadable)) cos(double3);
double4 const_func __attribute__((overloadable)) cos(double4);
double8 const_func __attribute__((overloadable)) cos(double8);
double16 const_func __attribute__((overloadable)) cos(double16);
float const_func __attribute__((overloadable)) cos(float);
float2 const_func __attribute__((overloadable)) cos(float2);
float3 const_func __attribute__((overloadable)) cos(float3);
float4 const_func __attribute__((overloadable)) cos(float4);
float8 const_func __attribute__((overloadable)) cos(float8);
float16 const_func __attribute__((overloadable)) cos(float16);
//half const_func __attribute__((overloadable)) cos(half);
//half2 const_func __attribute__((overloadable)) cos(half2);
//half3 const_func __attribute__((overloadable)) cos(half3);
//half4 const_func __attribute__((overloadable)) cos(half4);
//half8 const_func __attribute__((overloadable)) cos(half8);
//half16 const_func __attribute__((overloadable)) cos(half16);

/**
 * Compute hyperbolic consine.
 */
float const_func __attribute__((overloadable)) cosh(float);
float2 const_func __attribute__((overloadable)) cosh(float2);
float3 const_func __attribute__((overloadable)) cosh(float3);
float4 const_func __attribute__((overloadable)) cosh(float4);
float8 const_func __attribute__((overloadable)) cosh(float8);
float16 const_func __attribute__((overloadable)) cosh(float16);
double const_func __attribute__((overloadable)) cosh(double);
double2 const_func __attribute__((overloadable)) cosh(double2);
double3 const_func __attribute__((overloadable)) cosh(double3);
double4 const_func __attribute__((overloadable)) cosh(double4);
double8 const_func __attribute__((overloadable)) cosh(double8);
double16 const_func __attribute__((overloadable)) cosh(double16);
//half const_func __attribute__((overloadable)) cosh(half);
//half2 const_func __attribute__((overloadable)) cosh(half2);
//half3 const_func __attribute__((overloadable)) cosh(half3);
//half4 const_func __attribute__((overloadable)) cosh(half4);
//half8 const_func __attribute__((overloadable)) cosh(half8);
//half16 const_func __attribute__((overloadable)) cosh(half16);

/**
 * Compute cos (PI * x).
 */
float const_func __attribute__((overloadable)) cospi(float x);
float2 const_func __attribute__((overloadable)) cospi(float2 x);
float3 const_func __attribute__((overloadable)) cospi(float3 x);
float4 const_func __attribute__((overloadable)) cospi(float4 x);
float8 const_func __attribute__((overloadable)) cospi(float8 x);
float16 const_func __attribute__((overloadable)) cospi(float16 x);
double const_func __attribute__((overloadable)) cospi(double x);
double2 const_func __attribute__((overloadable)) cospi(double2 x);
double3 const_func __attribute__((overloadable)) cospi(double3 x);
double4 const_func __attribute__((overloadable)) cospi(double4 x);
double8 const_func __attribute__((overloadable)) cospi(double8 x);
double16 const_func __attribute__((overloadable)) cospi(double16 x);
//half const_func __attribute__((overloadable)) cospi(half x);
//half2 const_func __attribute__((overloadable)) cospi(half2 x);
//half3 const_func __attribute__((overloadable)) cospi(half3 x);
//half4 const_func __attribute__((overloadable)) cospi(half4 x);
//half8 const_func __attribute__((overloadable)) cospi(half8 x);
//half16 const_func __attribute__((overloadable)) cospi(half16 x);

/**
 * Complementary error function.
 */
float const_func __attribute__((overloadable)) erfc(float);
float2 const_func __attribute__((overloadable)) erfc(float2);
float3 const_func __attribute__((overloadable)) erfc(float3);
float4 const_func __attribute__((overloadable)) erfc(float4);
float8 const_func __attribute__((overloadable)) erfc(float8);
float16 const_func __attribute__((overloadable)) erfc(float16);
double const_func __attribute__((overloadable)) erfc(double);
double2 const_func __attribute__((overloadable)) erfc(double2);
double3 const_func __attribute__((overloadable)) erfc(double3);
```

```
double4 const_func __attribute__((overloadable)) erfc(double4);
double8 const_func __attribute__((overloadable)) erfc(double8);
double16 const_func __attribute__((overloadable)) erfc(double16);
//half const_func __attribute__((overloadable)) erfc(half);
//half2 const_func __attribute__((overloadable)) erfc(half2);
//half3 const_func __attribute__((overloadable)) erfc(half3);
//half4 const_func __attribute__((overloadable)) erfc(half4);
//half8 const_func __attribute__((overloadable)) erfc(half8);
//half16 const_func __attribute__((overloadable)) erfc(half16);

/**
 * Error function encountered in integrating the
 * normal distribution.
 */
float const_func __attribute__((overloadable)) erf(float);
float2 const_func __attribute__((overloadable)) erf(float2);
float3 const_func __attribute__((overloadable)) erf(float3);
float4 const_func __attribute__((overloadable)) erf(float4);
float8 const_func __attribute__((overloadable)) erf(float8);
float16 const_func __attribute__((overloadable)) erf(float16);
double const_func __attribute__((overloadable)) erf(double);
double2 const_func __attribute__((overloadable)) erf(double2);
double3 const_func __attribute__((overloadable)) erf(double3);
double4 const_func __attribute__((overloadable)) erf(double4);
double8 const_func __attribute__((overloadable)) erf(double8);
double16 const_func __attribute__((overloadable)) erf(double16);
//half const_func __attribute__((overloadable)) erf(half);
//half2 const_func __attribute__((overloadable)) erf(half2);
//half3 const_func __attribute__((overloadable)) erf(half3);
//half4 const_func __attribute__((overloadable)) erf(half4);
//half8 const_func __attribute__((overloadable)) erf(half8);
//half16 const_func __attribute__((overloadable)) erf(half16);

/**
 * Compute the base- e exponential of x.
 */
float const_func __attribute__((overloadable)) exp(float
x);
float2 const_func __attribute__((overloadable)) exp(float2
x);
float3 const_func __attribute__((overloadable)) exp(float3
x);
float4 const_func __attribute__((overloadable)) exp(float4
x);
float8 const_func __attribute__((overloadable)) exp(float8
x);
float16 const_func __attribute__((overloadable)) exp(float16
x);
double const_func __attribute__((overloadable)) exp(double
x);
double2 const_func __attribute__((overloadable)) exp(double2
x);
double3 const_func __attribute__((overloadable)) exp(double3
x);
double4 const_func __attribute__((overloadable)) exp(double4
x);
double8 const_func __attribute__((overloadable)) exp(double8
x);
double16 const_func __attribute__((overloadable)) exp(double16
x);
//half const_func __attribute__((overloadable)) exp(half x);
//half2 const_func __attribute__((overloadable)) exp(half2 x);
//half3 const_func __attribute__((overloadable)) exp(half3 x);
//half4 const_func __attribute__((overloadable)) exp(half4 x);
//half8 const_func __attribute__((overloadable)) exp(half8 x);
//half16 const_func __attribute__((overloadable)) exp(half16 x);

/**
 * Exponential base 2 function.
 */
float const_func __attribute__((overloadable)) exp2(float);
float2 const_func __attribute__((overloadable)) exp2(float2);
float3 const_func __attribute__((overloadable)) exp2(float3);
float4 const_func __attribute__((overloadable)) exp2(float4);
float8 const_func __attribute__((overloadable)) exp2(float8);
float16 const_func __attribute__((overloadable)) exp2(float16);
double const_func __attribute__((overloadable)) exp2(double);
double2 const_func __attribute__((overloadable)) exp2(double2);
double3 const_func __attribute__((overloadable)) exp2(double3);
double4 const_func __attribute__((overloadable)) exp2(double4);
double8 const_func __attribute__((overloadable)) exp2(double8);
double16 const_func __attribute__((overloadable)) exp2(double16);
//half const_func __attribute__((overloadable)) exp2(half);
//half2 const_func __attribute__((overloadable)) exp2(half2);
//half3 const_func __attribute__((overloadable)) exp2(half3);
//half4 const_func __attribute__((overloadable)) exp2(half4);
//half8 const_func __attribute__((overloadable)) exp2(half8);
```

```
//half16 const_func __attribute__((overloadable)) exp2(half16);

/**
 * Exponential base 10 function.
 */
float const_func __attribute__((overloadable)) exp10(float);
float2 const_func __attribute__((overloadable)) exp10(float2);
float3 const_func __attribute__((overloadable)) exp10(float3);
float4 const_func __attribute__((overloadable)) exp10(float4);
float8 const_func __attribute__((overloadable)) exp10(float8);
float16 const_func __attribute__((overloadable)) exp10(float16);
double const_func __attribute__((overloadable)) exp10(double);
double2 const_func __attribute__((overloadable)) exp10(double2);
double3 const_func __attribute__((overloadable)) exp10(double3);
double4 const_func __attribute__((overloadable)) exp10(double4);
double8 const_func __attribute__((overloadable)) exp10(double8);
double16 const_func __attribute__((overloadable)) exp10(double16);
//half const_func __attribute__((overloadable)) exp10(half);
//half2 const_func __attribute__((overloadable)) exp10(half2);
//half3 const_func __attribute__((overloadable)) exp10(half3);
//half4 const_func __attribute__((overloadable)) exp10(half4);
//half8 const_func __attribute__((overloadable)) exp10(half8);
//half16 const_func __attribute__((overloadable)) exp10(half16);

/**
 * Compute e^x- 1.0.
 */
float const_func __attribute__((overloadable)) expm1(float x);
float2 const_func __attribute__((overloadable)) expm1(float2 x);
float3 const_func __attribute__((overloadable)) expm1(float3 x);
float4 const_func __attribute__((overloadable)) expm1(float4 x);
float8 const_func __attribute__((overloadable)) expm1(float8 x);
float16 const_func __attribute__((overloadable)) expm1(float16 x);
double const_func __attribute__((overloadable)) expm1(double x);
double2 const_func __attribute__((overloadable)) expm1(double2 x);
double3 const_func __attribute__((overloadable)) expm1(double3 x);
double4 const_func __attribute__((overloadable)) expm1(double4 x);
double8 const_func __attribute__((overloadable)) expm1(double8 x);
double16 const_func __attribute__((overloadable)) expm1(double16 x);
//half const_func __attribute__((overloadable)) expm1(half x);
//half2 const_func __attribute__((overloadable)) expm1(half2 x);
//half3 const_func __attribute__((overloadable)) expm1(half3 x);
//half4 const_func __attribute__((overloadable)) expm1(half4 x);
//half8 const_func __attribute__((overloadable)) expm1(half8 x);
//half16 const_func __attribute__((overloadable)) expm1(half16 x);

/**
 * Compute absolute value of a floating-point number.
 */
float const_func __attribute__((overloadable)) fabs(float);
float2 const_func __attribute__((overloadable)) fabs(float2);
float3 const_func __attribute__((overloadable)) fabs(float3);
float4 const_func __attribute__((overloadable)) fabs(float4);
float8 const_func __attribute__((overloadable)) fabs(float8);
float16 const_func __attribute__((overloadable)) fabs(float16);
double const_func __attribute__((overloadable)) fabs(double);
double2 const_func __attribute__((overloadable)) fabs(double2);
double3 const_func __attribute__((overloadable)) fabs(double3);
double4 const_func __attribute__((overloadable)) fabs(double4);
double8 const_func __attribute__((overloadable)) fabs(double8);
double16 const_func __attribute__((overloadable)) fabs(double16);
//half const_func __attribute__((overloadable)) fabs(half);
//half2 const_func __attribute__((overloadable)) fabs(half2);
//half3 const_func __attribute__((overloadable)) fabs(half3);
//half4 const_func __attribute__((overloadable)) fabs(half4);
//half8 const_func __attribute__((overloadable)) fabs(half8);
//half16 const_func __attribute__((overloadable)) fabs(half16);

/**
 * x - y if x > y, +0 if x is less than or equal to y.
 */
float const_func __attribute__((overloadable)) fdim(float x, float y);
float2 const_func __attribute__((overloadable)) fdim(float2 x, float2 y);
```

```
float3 const_func __attribute__((overloadable)) fdim(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) fdim(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) fdim(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) fdim(float16 x, float16 y);
double const_func __attribute__((overloadable)) fdim(double x, double y);
double2 const_func __attribute__((overloadable)) fdim(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) fdim(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) fdim(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) fdim(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) fdim(double16 x, double16 y);
//half const_func __attribute__((overloadable)) fdim(half x, half y);
//half2 const_func __attribute__((overloadable)) fdim(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) fdim(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) fdim(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) fdim(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) fdim(half16 x, half16 y);

/**
 * Round to integral value using the round to -ve
 * infinity rounding mode.
 */
float const_func __attribute__((overloadable)) floor(float);
float2 const_func __attribute__((overloadable)) floor(float2);
float3 const_func __attribute__((overloadable)) floor(float3);
float4 const_func __attribute__((overloadable)) floor(float4);
float8 const_func __attribute__((overloadable)) floor(float8);
float16 const_func __attribute__((overloadable)) floor(float16);
double const_func __attribute__((overloadable)) floor(double);
double2 const_func __attribute__((overloadable)) floor(double2);
double3 const_func __attribute__((overloadable)) floor(double3);
double4 const_func __attribute__((overloadable)) floor(double4);
double8 const_func __attribute__((overloadable)) floor(double8);
double16 const_func __attribute__((overloadable)) floor(double16);
//half const_func __attribute__((overloadable)) floor(half);
//half2 const_func __attribute__((overloadable)) floor(half2);
//half3 const_func __attribute__((overloadable)) floor(half3);
//half4 const_func __attribute__((overloadable)) floor(half4);
//half8 const_func __attribute__((overloadable)) floor(half8);
//half16 const_func __attribute__((overloadable)) floor(half16);

/**
 * Returns the correctly rounded floating-point
 * representation of the sum of c with the infinitely
 * precise product of a and b. Rounding of
 * intermediate products shall not occur. Edge case
 * behavior is per the IEEE 754-2008 standard.
 */
float const_func __attribute__((overloadable)) fma(float a, float b, float c);
float2 const_func __attribute__((overloadable)) fma(float2 a, float2 b, float2 c);
float3 const_func __attribute__((overloadable)) fma(float3 a, float3 b, float3 c);
float4 const_func __attribute__((overloadable)) fma(float4 a, float4 b, float4 c);
float8 const_func __attribute__((overloadable)) fma(float8 a, float8 b, float8 c);
float16 const_func __attribute__((overloadable)) fma(float16 a, float16 b, float16 c);
double const_func __attribute__((overloadable)) fma(double a, double b, double c);
double2 const_func __attribute__((overloadable)) fma(double2 a, double2 b, double2 c);
double3 const_func __attribute__((overloadable)) fma(double3 a, double3 b, double3 c);
double4 const_func __attribute__((overloadable)) fma(double4 a, double4 b, double4 c);
double8 const_func __attribute__((overloadable)) fma(double8 a, double8 b, double8 c);
double16 const_func __attribute__((overloadable)) fma(double16 a, double16 b, double16 c);
//half const_func __attribute__((overloadable)) fma(half a, half b, half c);
//half2 const_func __attribute__((overloadable)) fma(half2 a, half2 b, half2 c);
//half3 const_func __attribute__((overloadable)) fma(half3 a, half3 b, half3 c);
//half4 const_func __attribute__((overloadable)) fma(half4 a, half4 b, half4 c);
//half8 const_func __attribute__((overloadable)) fma(half8 a, half8 b, half8 c);
//half16 const_func __attribute__((overloadable)) fma(half16 a, half16 b, half16 c);
```

```

/**
 * Returns y if x < y, otherwise it returns x. If one
 * argument is a NaN, fmax() returns the other
 * argument. If both arguments are NaNs, fmax()
 * returns a NaN.
 */
float const_func __attribute__((overloadable)) fmax(float x, float y);
float2 const_func __attribute__((overloadable)) fmax(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) fmax(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) fmax(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) fmax(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) fmax(float16 x, float16 y);
float2 const_func __attribute__((overloadable)) fmax(float2 x, float y);
float3 const_func __attribute__((overloadable)) fmax(float3 x, float y);
float4 const_func __attribute__((overloadable)) fmax(float4 x, float y);
float8 const_func __attribute__((overloadable)) fmax(float8 x, float y);
float16 const_func __attribute__((overloadable)) fmax(float16 x, float y);
double const_func __attribute__((overloadable)) fmax(double x, double y);
double2 const_func __attribute__((overloadable)) fmax(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) fmax(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) fmax(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) fmax(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) fmax(double16 x, double16 y);
double2 const_func __attribute__((overloadable)) fmax(double2 x, double y);
double3 const_func __attribute__((overloadable)) fmax(double3 x, double y);
double4 const_func __attribute__((overloadable)) fmax(double4 x, double y);
double8 const_func __attribute__((overloadable)) fmax(double8 x, double y);
double16 const_func __attribute__((overloadable)) fmax(double16 x, double y);
//half const_func __attribute__((overloadable)) fmax(half x, half y);
//half2 const_func __attribute__((overloadable)) fmax(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) fmax(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) fmax(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) fmax(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) fmax(half16 x, half16 y);
//half2 const_func __attribute__((overloadable)) fmax(half2 x, half y);
//half3 const_func __attribute__((overloadable)) fmax(half3 x, half y);
//half4 const_func __attribute__((overloadable)) fmax(half4 x, half y);
//half8 const_func __attribute__((overloadable)) fmax(half8 x, half y);
//half16 const_func __attribute__((overloadable)) fmax(half16 x, half y);

/**
 * Returns y if y < x, otherwise it returns x. If one
 * argument is a NaN, fmin() returns the other
 * argument. If both arguments are NaNs, fmin()
 * returns a NaN.
 */
float const_func __attribute__((overloadable)) fmin(float x, float y);
float2 const_func __attribute__((overloadable)) fmin(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) fmin(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) fmin(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) fmin(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) fmin(float16 x, float16 y);
float2 const_func __attribute__((overloadable)) fmin(float2 x, float y);
float3 const_func __attribute__((overloadable)) fmin(float3 x, float y);
float4 const_func __attribute__((overloadable)) fmin(float4 x, float y);
```

```
float8 const_func __attribute__((overloadable)) fmin(float8 x, float y);
float16 const_func __attribute__((overloadable)) fmin(float16 x, float y);
double const_func __attribute__((overloadable)) fmin(double x, double y);
double2 const_func __attribute__((overloadable)) fmin(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) fmin(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) fmin(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) fmin(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) fmin(double16 x, double16 y);
double2 const_func __attribute__((overloadable)) fmin(double2 x, double y);
double3 const_func __attribute__((overloadable)) fmin(double3 x, double y);
double4 const_func __attribute__((overloadable)) fmin(double4 x, double y);
double8 const_func __attribute__((overloadable)) fmin(double8 x, double y);
double16 const_func __attribute__((overloadable)) fmin(double16 x, double y);
//half const_func __attribute__((overloadable)) fmin(half x, half y);
//half2 const_func __attribute__((overloadable)) fmin(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) fmin(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) fmin(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) fmin(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) fmin(half16 x, half16 y);
//half2 const_func __attribute__((overloadable)) fmin(half2 x, half y);
//half3 const_func __attribute__((overloadable)) fmin(half3 x, half y);
//half4 const_func __attribute__((overloadable)) fmin(half4 x, half y);
//half8 const_func __attribute__((overloadable)) fmin(half8 x, half y);
//half16 const_func __attribute__((overloadable)) fmin(half16 x, half y);

/**
 * Modulus. Returns x - y * trunc (x/y).
 */
float const_func __attribute__((overloadable)) fmod(float x, float y);
float2 const_func __attribute__((overloadable)) fmod(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) fmod(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) fmod(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) fmod(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) fmod(float16 x, float16 y);
double const_func __attribute__((overloadable)) fmod(double x, double y);
double2 const_func __attribute__((overloadable)) fmod(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) fmod(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) fmod(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) fmod(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) fmod(double16 x, double16 y);
//half const_func __attribute__((overloadable)) fmod(half x, half y);
//half2 const_func __attribute__((overloadable)) fmod(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) fmod(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) fmod(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) fmod(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) fmod(half16 x, half16 y);

/**
 * Returns fmin( x - floor (x), 0x1.fffffep-1f ).
 * floor(x) is returned in iptr.
 */
float __attribute__((overloadable)) fract(float x, __global float * iptr);
float2 __attribute__((overloadable)) fract(float2 x, __global float2 * iptr);
float3 __attribute__((overloadable)) fract(float3 x, __global float3 * iptr);
float4 __attribute__((overloadable)) fract(float4 x, __global float4 * iptr);
float8 __attribute__((overloadable)) fract(float8 x, __global float8 * iptr);
float16 __attribute__((overloadable)) fract(float16 x, __global float16 *
```

```
iptr);
float __attribute__((overloadable)) fract(float x, __local float *
iptr);
float2 __attribute__((overloadable)) fract(float2 x, __local float2 *
iptr);
float3 __attribute__((overloadable)) fract(float3 x, __local float3 *
iptr);
float4 __attribute__((overloadable)) fract(float4 x, __local float4 *
iptr);
float8 __attribute__((overloadable)) fract(float8 x, __local float8 *
iptr);
float16 __attribute__((overloadable)) fract(float16 x, __local float16 *
iptr);
float __attribute__((overloadable)) fract(float x, __private float *
iptr);
float2 __attribute__((overloadable)) fract(float2 x, __private float2 *
iptr);
float3 __attribute__((overloadable)) fract(float3 x, __private float3 *
iptr);
float4 __attribute__((overloadable)) fract(float4 x, __private float4 *
iptr);
float8 __attribute__((overloadable)) fract(float8 x, __private float8 *
iptr);
float16 __attribute__((overloadable)) fract(float16 x, __private float16 *
iptr);
double __attribute__((overloadable)) fract(double x, __global double *
iptr);
double2 __attribute__((overloadable)) fract(double2 x, __global double2 *
iptr);
double3 __attribute__((overloadable)) fract(double3 x, __global double3 *
iptr);
double4 __attribute__((overloadable)) fract(double4 x, __global double4 *
iptr);
double8 __attribute__((overloadable)) fract(double8 x, __global double8 *
iptr);
double16 __attribute__((overloadable)) fract(double16 x, __global double16 *
iptr);
double __attribute__((overloadable)) fract(double x, __local double *
iptr);
double2 __attribute__((overloadable)) fract(double2 x, __local double2 *
iptr);
double3 __attribute__((overloadable)) fract(double3 x, __local double3 *
iptr);
double4 __attribute__((overloadable)) fract(double4 x, __local double4 *
iptr);
double8 __attribute__((overloadable)) fract(double8 x, __local double8 *
iptr);
double16 __attribute__((overloadable)) fract(double16 x, __local double16 *
iptr);
double __attribute__((overloadable)) fract(double x, __private double *
iptr);
double2 __attribute__((overloadable)) fract(double2 x, __private double2 *
iptr);
double3 __attribute__((overloadable)) fract(double3 x, __private double3 *
iptr);
double4 __attribute__((overloadable)) fract(double4 x, __private double4 *
iptr);
double8 __attribute__((overloadable)) fract(double8 x, __private double8 *
iptr);
double16 __attribute__((overloadable)) fract(double16 x, __private double16 *
iptr);
//half __attribute__((overloadable)) fract(half x, __global half *iptr);
//half2 __attribute__((overloadable)) fract(half2 x, __global half2 *iptr);
//half3 __attribute__((overloadable)) fract(half3 x, __global half3 *iptr);
//half4 __attribute__((overloadable)) fract(half4 x, __global half4 *iptr);
//half8 __attribute__((overloadable)) fract(half8 x, __global half8 *iptr);
//half16 __attribute__((overloadable)) fract(half16 x, __global half16 *iptr);
//half __attribute__((overloadable)) fract(half x, __local half *iptr);
//half2 __attribute__((overloadable)) fract(half2 x, __local half2 *iptr);
//half3 __attribute__((overloadable)) fract(half3 x, __local half3 *iptr);
//half4 __attribute__((overloadable)) fract(half4 x, __local half4 *iptr);
//half8 __attribute__((overloadable)) fract(half8 x, __local half8 *iptr);
//half16 __attribute__((overloadable)) fract(half16 x, __local half16 *iptr);
//half __attribute__((overloadable)) fract(half x, __private half *iptr);
//half2 __attribute__((overloadable)) fract(half2 x, __private half2 *iptr);
//half3 __attribute__((overloadable)) fract(half3 x, __private half3 *iptr);
//half4 __attribute__((overloadable)) fract(half4 x, __private half4 *iptr);
//half8 __attribute__((overloadable)) fract(half8 x, __private half8 *iptr);
//half16 __attribute__((overloadable)) fract(half16 x, __private half16 *iptr);


/**
 * Extract mantissa and exponent from x. For each
 * component the mantissa returned is a float with
 * magnitude in the interval [1/2, 1) or 0. Each
 * component of x equals mantissa returned * 2^exp.
 */
```

```
float __attribute__((overloadable)) frexp(float x, __global int *exp);
float2 __attribute__((overloadable)) frexp(float2 x, __global int2 *exp);
float3 __attribute__((overloadable)) frexp(float3 x, __global int3 *exp);
float4 __attribute__((overloadable)) frexp(float4 x, __global int4 *exp);
float8 __attribute__((overloadable)) frexp(float8 x, __global int8 *exp);
float16 __attribute__((overloadable)) frexp(float16 x, __global int16 *exp);
float __attribute__((overloadable)) frexp(float x, __local int *exp);
float2 __attribute__((overloadable)) frexp(float2 x, __local int2 *exp);
float3 __attribute__((overloadable)) frexp(float3 x, __local int3 *exp);
float4 __attribute__((overloadable)) frexp(float4 x, __local int4 *exp);
float8 __attribute__((overloadable)) frexp(float8 x, __local int8 *exp);
float16 __attribute__((overloadable)) frexp(float16 x, __local int16 *exp);
float __attribute__((overloadable)) frexp(float x, __private int *exp);
float2 __attribute__((overloadable)) frexp(float2 x, __private int2 *exp);
float3 __attribute__((overloadable)) frexp(float3 x, __private int3 *exp);
float4 __attribute__((overloadable)) frexp(float4 x, __private int4 *exp);
float8 __attribute__((overloadable)) frexp(float8 x, __private int8 *exp);
float16 __attribute__((overloadable)) frexp(float16 x, __private int16 *exp);
double __attribute__((overloadable)) frexp(double x, __global int *exp);
double2 __attribute__((overloadable)) frexp(double2 x, __global int2 *exp);
double3 __attribute__((overloadable)) frexp(double3 x, __global int3 *exp);
double4 __attribute__((overloadable)) frexp(double4 x, __global int4 *exp);
double8 __attribute__((overloadable)) frexp(double8 x, __global int8 *exp);
double16 __attribute__((overloadable)) frexp(double16 x, __global int16 *exp);
double __attribute__((overloadable)) frexp(double x, __local int *exp);
double2 __attribute__((overloadable)) frexp(double2 x, __local int2 *exp);
double3 __attribute__((overloadable)) frexp(double3 x, __local int3 *exp);
double4 __attribute__((overloadable)) frexp(double4 x, __local int4 *exp);
double8 __attribute__((overloadable)) frexp(double8 x, __local int8 *exp);
double16 __attribute__((overloadable)) frexp(double16 x, __local int16 *exp);
double __attribute__((overloadable)) frexp(double x, __private int *exp);
double2 __attribute__((overloadable)) frexp(double2 x, __private int2 *exp);
double3 __attribute__((overloadable)) frexp(double3 x, __private int3 *exp);
double4 __attribute__((overloadable)) frexp(double4 x, __private int4 *exp);
double8 __attribute__((overloadable)) frexp(double8 x, __private int8 *exp);
double16 __attribute__((overloadable)) frexp(double16 x, __private int16 *exp);
//half __attribute__((overloadable)) frexp(half x, __global int *exp);
//half2 __attribute__((overloadable)) frexp(half2 x, __global int2 *exp);
//half3 __attribute__((overloadable)) frexp(half3 x, __global int3 *exp);
//half4 __attribute__((overloadable)) frexp(half4 x, __global int4 *exp);
//half8 __attribute__((overloadable)) frexp(half8 x, __global int8 *exp);
//half16 __attribute__((overloadable)) frexp(half16 x, __global int16 *exp);
//half __attribute__((overloadable)) frexp(half x, __local int *exp);
//half2 __attribute__((overloadable)) frexp(half2 x, __local int2 *exp);
//half3 __attribute__((overloadable)) frexp(half3 x, __local int3 *exp);
//half4 __attribute__((overloadable)) frexp(half4 x, __local int4 *exp);
//half8 __attribute__((overloadable)) frexp(half8 x, __local int8 *exp);
//half16 __attribute__((overloadable)) frexp(half16 x, __local int16 *exp);
//half __attribute__((overloadable)) frexp(half x, __private int *exp);
//half2 __attribute__((overloadable)) frexp(half2 x, __private int2 *exp);
//half3 __attribute__((overloadable)) frexp(half3 x, __private int3 *exp);
```

```
//half4 __attribute__((overloadable)) frexp(half4 x, __private int4 *exp);
//half8 __attribute__((overloadable)) frexp(half8 x, __private int8 *exp);
//half16 __attribute__((overloadable)) frexp(half16 x, __private int16 *exp);

/**
 * Compute the value of the square root of x^2+ y^2
 * without undue overflow or underflow.
 */
float const_func __attribute__((overloadable)) hypot(float x, float y);
float2 const_func __attribute__((overloadable)) hypot(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) hypot(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) hypot(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) hypot(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) hypot(float16 x, float16 y);
double const_func __attribute__((overloadable)) hypot(double x, double y);
double2 const_func __attribute__((overloadable)) hypot(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) hypot(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) hypot(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) hypot(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) hypot(double16 x, double16 y);
//half const_func __attribute__((overloadable)) hypot(half x, half y);
//half2 const_func __attribute__((overloadable)) hypot(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) hypot(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) hypot(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) hypot(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) hypot(half16 x, half16 y);

/**
 * Return the exponent as an integer value.
 */
int const_func __attribute__((overloadable)) ilogb(float x);
int2 const_func __attribute__((overloadable)) ilogb(float2 x);
int3 const_func __attribute__((overloadable)) ilogb(float3 x);
int4 const_func __attribute__((overloadable)) ilogb(float4 x);
int8 const_func __attribute__((overloadable)) ilogb(float8 x);
int16 const_func __attribute__((overloadable)) ilogb(float16 x);
int const_func __attribute__((overloadable)) ilogb(double x);
int2 const_func __attribute__((overloadable)) ilogb(double2 x);
int3 const_func __attribute__((overloadable)) ilogb(double3 x);
int4 const_func __attribute__((overloadable)) ilogb(double4 x);
int8 const_func __attribute__((overloadable)) ilogb(double8 x);
int16 const_func __attribute__((overloadable)) ilogb(double16 x);
//int const_func __attribute__((overloadable)) ilogb(half x);
//int2 const_func __attribute__((overloadable)) ilogb(half2 x);
//int3 const_func __attribute__((overloadable)) ilogb(half3 x);
//int4 const_func __attribute__((overloadable)) ilogb(half4 x);
//int8 const_func __attribute__((overloadable)) ilogb(half8 x);
//int16 const_func __attribute__((overloadable)) ilogb(half16 x);

/**
 * Multiply x by 2 to the power n.
 */
float const_func __attribute__((overloadable)) ldexp(float x, int n);
float2 const_func __attribute__((overloadable)) ldexp(float2 x, int2 n);
float3 const_func __attribute__((overloadable)) ldexp(float3 x, int3 n);
float4 const_func __attribute__((overloadable)) ldexp(float4 x, int4 n);
float8 const_func __attribute__((overloadable)) ldexp(float8 x, int8 n);
float16 const_func __attribute__((overloadable)) ldexp(float16
```

```
x, int16 n);
float2 const_func __attribute__((overloadable)) ldexp(float2 x, int n);
float3 const_func __attribute__((overloadable)) ldexp(float3 x, int n);
float4 const_func __attribute__((overloadable)) ldexp(float4 x, int n);
float8 const_func __attribute__((overloadable)) ldexp(float8 x, int n);
float16 const_func __attribute__((overloadable)) ldexp(float16 x, int n);
double const_func __attribute__((overloadable)) ldexp(double x, int n);
double2 const_func __attribute__((overloadable)) ldexp(double2 x, int2 n);
double3 const_func __attribute__((overloadable)) ldexp(double3 x, int3 n);
double4 const_func __attribute__((overloadable)) ldexp(double4 x, int4 n);
double8 const_func __attribute__((overloadable)) ldexp(double8 x, int8 n);
double16 const_func __attribute__((overloadable)) ldexp(double16 x, int16 n);
double2 const_func __attribute__((overloadable)) ldexp(double2 x, int n);
double3 const_func __attribute__((overloadable)) ldexp(double3 x, int n);
double4 const_func __attribute__((overloadable)) ldexp(double4 x, int n);
double8 const_func __attribute__((overloadable)) ldexp(double8 x, int n);
double16 const_func __attribute__((overloadable)) ldexp(double16 x, int n);
//half const_func __attribute__((overloadable)) ldexp(half x, int n);
//half2 const_func __attribute__((overloadable)) ldexp(half2 x, int2 n);
//half3 const_func __attribute__((overloadable)) ldexp(half3 x, int3 n);
//half4 const_func __attribute__((overloadable)) ldexp(half4 x, int4 n);
//half8 const_func __attribute__((overloadable)) ldexp(half8 x, int8 n);
//half16 const_func __attribute__((overloadable)) ldexp(half16 x, int16 n);
//half const_func __attribute__((overloadable)) ldexp(half x, int n);
//half2 const_func __attribute__((overloadable)) ldexp(half2 x, int n);
//half3 const_func __attribute__((overloadable)) ldexp(half3 x, int n);
//half4 const_func __attribute__((overloadable)) ldexp(half4 x, int n);
//half8 const_func __attribute__((overloadable)) ldexp(half8 x, int n);
//half16 const_func __attribute__((overloadable)) ldexp(half16 x, int n);

/**
 * Log gamma function. Returns the natural
 * logarithm of the absolute value of the gamma
 * function. The sign of the gamma function is
 * returned in the signp argument of lgamma_r.
 */
float const_func __attribute__((overloadable)) lgamma(float x);
float2 const_func __attribute__((overloadable)) lgamma(float2 x);
float3 const_func __attribute__((overloadable)) lgamma(float3 x);
float4 const_func __attribute__((overloadable)) lgamma(float4 x);
float8 const_func __attribute__((overloadable)) lgamma(float8 x);
float16 const_func __attribute__((overloadable)) lgamma(float16 x);
double const_func __attribute__((overloadable)) lgamma(double x);
double2 const_func __attribute__((overloadable)) lgamma(double2 x);
double3 const_func __attribute__((overloadable)) lgamma(double3 x);
double4 const_func __attribute__((overloadable)) lgamma(double4 x);
double8 const_func __attribute__((overloadable)) lgamma(double8 x);
double16 const_func __attribute__((overloadable)) lgamma(double16 x);
//half const_func __attribute__((overloadable)) lgamma(half x);
//half2 const_func __attribute__((overloadable)) lgamma(half2 x);
//half3 const_func __attribute__((overloadable)) lgamma(half3 x);
//half4 const_func __attribute__((overloadable)) lgamma(half4 x);
//half8 const_func __attribute__((overloadable)) lgamma(half8 x);
//half16 const_func __attribute__((overloadable)) lgamma(half16 x);

float __attribute__((overloadable)) lgamma_r(float x, __global int * signp);
float2 __attribute__((overloadable)) lgamma_r(float2 x, __global int2 * signp);
```

```
float3 __attribute__((overloadable)) lgamma_r(float3 x, __global int3 *
signp);
float4 __attribute__((overloadable)) lgamma_r(float4 x, __global int4 *
signp);
float8 __attribute__((overloadable)) lgamma_r(float8 x, __global int8 *
signp);
float16 __attribute__((overloadable)) lgamma_r(float16 x, __global int16 *
signp);
float __attribute__((overloadable)) lgamma_r(float x, __local int *
signp);
float2 __attribute__((overloadable)) lgamma_r(float2 x, __local int2 *
signp);
float3 __attribute__((overloadable)) lgamma_r(float3 x, __local int3 *
signp);
float4 __attribute__((overloadable)) lgamma_r(float4 x, __local int4 *
signp);
float8 __attribute__((overloadable)) lgamma_r(float8 x, __local int8 *
signp);
float16 __attribute__((overloadable)) lgamma_r(float16 x, __local int16 *
signp);
float __attribute__((overloadable)) lgamma_r(float x, __private int *
signp);
float2 __attribute__((overloadable)) lgamma_r(float2 x, __private int2 *
signp);
float3 __attribute__((overloadable)) lgamma_r(float3 x, __private int3 *
signp);
float4 __attribute__((overloadable)) lgamma_r(float4 x, __private int4 *
signp);
float8 __attribute__((overloadable)) lgamma_r(float8 x, __private int8 *
signp);
float16 __attribute__((overloadable)) lgamma_r(float16 x, __private int16 *
signp);
double __attribute__((overloadable)) lgamma_r(double x, __global int *
signp);
double2 __attribute__((overloadable)) lgamma_r(double2 x, __global int2 *
signp);
double3 __attribute__((overloadable)) lgamma_r(double3 x, __global int3 *
signp);
double4 __attribute__((overloadable)) lgamma_r(double4 x, __global int4 *
signp);
double8 __attribute__((overloadable)) lgamma_r(double8 x, __global int8 *
signp);
double16 __attribute__((overloadable)) lgamma_r(double16 x, __global int16 *
signp);
double __attribute__((overloadable)) lgamma_r(double x, __local int *
signp);
double2 __attribute__((overloadable)) lgamma_r(double2 x, __local int2 *
signp);
double3 __attribute__((overloadable)) lgamma_r(double3 x, __local int3 *
signp);
double4 __attribute__((overloadable)) lgamma_r(double4 x, __local int4 *
signp);
double8 __attribute__((overloadable)) lgamma_r(double8 x, __local int8 *
signp);
double16 __attribute__((overloadable)) lgamma_r(double16 x, __local int16 *
signp);
double __attribute__((overloadable)) lgamma_r(double x, __private int *
signp);
double2 __attribute__((overloadable)) lgamma_r(double2 x, __private int2 *
signp);
double3 __attribute__((overloadable)) lgamma_r(double3 x, __private int3 *
signp);
double4 __attribute__((overloadable)) lgamma_r(double4 x, __private int4 *
signp);
double8 __attribute__((overloadable)) lgamma_r(double8 x, __private int8 *
signp);
double16 __attribute__((overloadable)) lgamma_r(double16 x, __private int16 *
signp);
//half __attribute__((overloadable)) lgamma_r(half x, __global int *signp);
//half2 __attribute__((overloadable)) lgamma_r(half2 x, __global int2 *signp);
//half3 __attribute__((overloadable)) lgamma_r(half3 x, __global int3 *signp);
//half4 __attribute__((overloadable)) lgamma_r(half4 x, __global int4 *signp);
//half8 __attribute__((overloadable)) lgamma_r(half8 x, __global int8 *signp);
//half16 __attribute__((overloadable)) lgamma_r(half16 x, __global int16 *signp);
//half __attribute__((overloadable)) lgamma_r(half x, __local int *signp);
//half2 __attribute__((overloadable)) lgamma_r(half2 x, __local int2 *signp);
//half3 __attribute__((overloadable)) lgamma_r(half3 x, __local int3 *signp);
//half4 __attribute__((overloadable)) lgamma_r(half4 x, __local int4 *signp);
//half8 __attribute__((overloadable)) lgamma_r(half8 x, __local int8 *signp);
//half16 __attribute__((overloadable)) lgamma_r(half16 x, __local int16 *signp);
//half __attribute__((overloadable)) lgamma_r(half x, __private int *signp);
//half2 __attribute__((overloadable)) lgamma_r(half2 x, __private int2 *signp);
//half3 __attribute__((overloadable)) lgamma_r(half3 x, __private int3 *signp);
//half4 __attribute__((overloadable)) lgamma_r(half4 x, __private int4 *signp);
//half8 __attribute__((overloadable)) lgamma_r(half8 x, __private int8 *signp);
//half16 __attribute__((overloadable)) lgamma_r(half16 x, __private int16 *signp);
```

```
/**
 * Compute natural logarithm.
 */
float const_func __attribute__((overloadable)) log(float);
float2 const_func __attribute__((overloadable)) log(float2);
float3 const_func __attribute__((overloadable)) log(float3);
float4 const_func __attribute__((overloadable)) log(float4);
float8 const_func __attribute__((overloadable)) log(float8);
float16 const_func __attribute__((overloadable)) log(float16);
double const_func __attribute__((overloadable)) log(double);
double2 const_func __attribute__((overloadable)) log(double2);
double3 const_func __attribute__((overloadable)) log(double3);
double4 const_func __attribute__((overloadable)) log(double4);
double8 const_func __attribute__((overloadable)) log(double8);
double16 const_func __attribute__((overloadable)) log(double16);
//half const_func __attribute__((overloadable)) log(half);
//half2 const_func __attribute__((overloadable)) log(half2);
//half3 const_func __attribute__((overloadable)) log(half3);
//half4 const_func __attribute__((overloadable)) log(half4);
//half8 const_func __attribute__((overloadable)) log(half8);
//half16 const_func __attribute__((overloadable)) log(half16);

/**
 * Compute a base 2 logarithm.
 */
float const_func __attribute__((overloadable)) log2(float);
float2 const_func __attribute__((overloadable)) log2(float2);
float3 const_func __attribute__((overloadable)) log2(float3);
float4 const_func __attribute__((overloadable)) log2(float4);
float8 const_func __attribute__((overloadable)) log2(float8);
float16 const_func __attribute__((overloadable)) log2(float16);
double const_func __attribute__((overloadable)) log2(double);
double2 const_func __attribute__((overloadable)) log2(double2);
double3 const_func __attribute__((overloadable)) log2(double3);
double4 const_func __attribute__((overloadable)) log2(double4);
double8 const_func __attribute__((overloadable)) log2(double8);
double16 const_func __attribute__((overloadable)) log2(double16);
//half const_func __attribute__((overloadable)) log2(half);
//half2 const_func __attribute__((overloadable)) log2(half2);
//half3 const_func __attribute__((overloadable)) log2(half3);
//half4 const_func __attribute__((overloadable)) log2(half4);
//half8 const_func __attribute__((overloadable)) log2(half8);
//half16 const_func __attribute__((overloadable)) log2(half16);

/**
 * Compute a base 10 logarithm.
 */
float const_func __attribute__((overloadable)) log10(float);
float2 const_func __attribute__((overloadable)) log10(float2);
float3 const_func __attribute__((overloadable)) log10(float3);
float4 const_func __attribute__((overloadable)) log10(float4);
float8 const_func __attribute__((overloadable)) log10(float8);
float16 const_func __attribute__((overloadable)) log10(float16);
double const_func __attribute__((overloadable)) log10(double);
double2 const_func __attribute__((overloadable)) log10(double2);
double3 const_func __attribute__((overloadable)) log10(double3);
double4 const_func __attribute__((overloadable)) log10(double4);
double8 const_func __attribute__((overloadable)) log10(double8);
double16 const_func __attribute__((overloadable)) log10(double16);
//half const_func __attribute__((overloadable)) log10(half);
//half2 const_func __attribute__((overloadable)) log10(half2);
//half3 const_func __attribute__((overloadable)) log10(half3);
//half4 const_func __attribute__((overloadable)) log10(half4);
//half8 const_func __attribute__((overloadable)) log10(half8);
//half16 const_func __attribute__((overloadable)) log10(half16);

/**
 * Compute a base e logarithm of (1.0 + x).
 */
float const_func __attribute__((overloadable)) log1p(float
x);
float2 const_func __attribute__((overloadable)) log1p(float2
x);
float3 const_func __attribute__((overloadable)) log1p(float3
x);
float4 const_func __attribute__((overloadable)) log1p(float4
x);
float8 const_func __attribute__((overloadable)) log1p(float8
x);
float16 const_func __attribute__((overloadable)) log1p(float16
x);
double const_func __attribute__((overloadable)) log1p(double
x);
double2 const_func __attribute__((overloadable)) log1p(double2
x);
double3 const_func __attribute__((overloadable)) log1p(double3
x);
```

```
double4 const_func __attribute__((overloadable)) log1p(double4 x);
double8 const_func __attribute__((overloadable)) log1p(double8 x);
double16 const_func __attribute__((overloadable)) log1p(double16 x);
//half const_func __attribute__((overloadable)) log1p(half x);
//half2 const_func __attribute__((overloadable)) log1p(half2 x);
//half3 const_func __attribute__((overloadable)) log1p(half3 x);
//half4 const_func __attribute__((overloadable)) log1p(half4 x);
//half8 const_func __attribute__((overloadable)) log1p(half8 x);
//half16 const_func __attribute__((overloadable)) log1p(half16 x);

/**
 * Compute the exponent of x, which is the integral
 * part of logr | x |.
 */
float const_func __attribute__((overloadable)) logb(float x);
float2 const_func __attribute__((overloadable)) logb(float2 x);
float3 const_func __attribute__((overloadable)) logb(float3 x);
float4 const_func __attribute__((overloadable)) logb(float4 x);
float8 const_func __attribute__((overloadable)) logb(float8 x);
float16 const_func __attribute__((overloadable)) logb(float16 x);
double const_func __attribute__((overloadable)) logb(double x);
double2 const_func __attribute__((overloadable)) logb(double2 x);
double3 const_func __attribute__((overloadable)) logb(double3 x);
double4 const_func __attribute__((overloadable)) logb(double4 x);
double8 const_func __attribute__((overloadable)) logb(double8 x);
double16 const_func __attribute__((overloadable)) logb(double16 x);
//half const_func __attribute__((overloadable)) logb(half x);
//half2 const_func __attribute__((overloadable)) logb(half2 x);
//half3 const_func __attribute__((overloadable)) logb(half3 x);
//half4 const_func __attribute__((overloadable)) logb(half4 x);
//half8 const_func __attribute__((overloadable)) logb(half8 x);
//half16 const_func __attribute__((overloadable)) logb(half16 x);

/**
 * mad approximates a * b + c. Whether or how the
 * product of a * b is rounded and how supernormal or
 * subnormal intermediate products are handled is not
 * defined. mad is intended to be used where speed is
 * preferred over accuracy.
 */
float const_func __attribute__((overloadable)) mad(float a, float b, float c);
float2 const_func __attribute__((overloadable)) mad(float2 a, float2 b, float2 c);
float3 const_func __attribute__((overloadable)) mad(float3 a, float3 b, float3 c);
float4 const_func __attribute__((overloadable)) mad(float4 a, float4 b, float4 c);
float8 const_func __attribute__((overloadable)) mad(float8 a, float8 b, float8 c);
float16 const_func __attribute__((overloadable)) mad(float16 a, float16 b, float16 c);
double const_func __attribute__((overloadable)) mad(double a, double b, double c);
double2 const_func __attribute__((overloadable)) mad(double2 a, double2 b, double2 c);
double3 const_func __attribute__((overloadable)) mad(double3 a, double3 b, double3 c);
double4 const_func __attribute__((overloadable)) mad(double4 a, double4 b, double4 c);
double8 const_func __attribute__((overloadable)) mad(double8 a, double8 b, double8 c);
double16 const_func __attribute__((overloadable)) mad(double16 a, double16 b, double16 c);
//half const_func __attribute__((overloadable)) mad(half a, half b, half c);
//half2 const_func __attribute__((overloadable)) mad(half2 a, half2 b, half2 c);
//half3 const_func __attribute__((overloadable)) mad(half3 a, half3 b, half3 c);
//half4 const_func __attribute__((overloadable)) mad(half4 a, half4 b, half4 c);
//half8 const_func __attribute__((overloadable)) mad(half8 a, half8 b, half8 c);
//half16 const_func __attribute__((overloadable)) mad(half16 a, half16 b, half16 c);

/**
```

```
 * Returns x if | x | > | y |, y if | y | > | x |, otherwise
 * fmax(x, y).
 */
float const_func __attribute__((overloadable)) maxmag(float x, float y);
float2 const_func __attribute__((overloadable)) maxmag(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) maxmag(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) maxmag(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) maxmag(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) maxmag(float16 x, float16 y);
double const_func __attribute__((overloadable)) maxmag(double x, double y);
double2 const_func __attribute__((overloadable)) maxmag(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) maxmag(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) maxmag(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) maxmag(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) maxmag(double16 x, double16 y);
//half const_func __attribute__((overloadable)) maxmag(half x, half y);
//half2 const_func __attribute__((overloadable)) maxmag(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) maxmag(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) maxmag(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) maxmag(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) maxmag(half16 x, half16 y);

/**
 * Returns x if | x | < | y |, y if | y | < | x |, otherwise
 * fmin(x, y).
 */
float const_func __attribute__((overloadable)) minmag(float x, float y);
float2 const_func __attribute__((overloadable)) minmag(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) minmag(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) minmag(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) minmag(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) minmag(float16 x, float16 y);
double const_func __attribute__((overloadable)) minmag(double x, double y);
double2 const_func __attribute__((overloadable)) minmag(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) minmag(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) minmag(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) minmag(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) minmag(double16 x, double16 y);
//half const_func __attribute__((overloadable)) minmag(half x, half y);
//half2 const_func __attribute__((overloadable)) minmag(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) minmag(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) minmag(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) minmag(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) minmag(half16 x, half16 y);

/**
 * Decompose a floating-point number. The modf
 * function breaks the argument x into integral and
 * fractional parts, each of which has the same sign as
 * the argument. It stores the integral part in the object
 * pointed to by iptr.
 */
float __attribute__((overloadable)) modf(float x, __global float * iptr);
float2 __attribute__((overloadable)) modf(float2 x, __global float2 * iptr);
float3 __attribute__((overloadable)) modf(float3 x, __global float3 * iptr);
float4 __attribute__((overloadable)) modf(float4 x, __global float4 * iptr);
float8 __attribute__((overloadable)) modf(float8 x, __global float8 * iptr);
float16 __attribute__((overloadable)) modf(float16 x, __global float16 *
```

```
iptr);
float __attribute__((overloadable)) modf(float x, __local float *
iptr);
float2 __attribute__((overloadable)) modf(float2 x, __local float2 *
iptr);
float3 __attribute__((overloadable)) modf(float3 x, __local float3 *
iptr);
float4 __attribute__((overloadable)) modf(float4 x, __local float4 *
iptr);
float8 __attribute__((overloadable)) modf(float8 x, __local float8 *
iptr);
float16 __attribute__((overloadable)) modf(float16 x, __local float16 *
iptr);
float __attribute__((overloadable)) modf(float x, __private float *
iptr);
float2 __attribute__((overloadable)) modf(float2 x, __private float2 *
iptr);
float3 __attribute__((overloadable)) modf(float3 x, __private float3 *
iptr);
float4 __attribute__((overloadable)) modf(float4 x, __private float4 *
iptr);
float8 __attribute__((overloadable)) modf(float8 x, __private float8 *
iptr);
float16 __attribute__((overloadable)) modf(float16 x, __private float16 *
iptr);
double __attribute__((overloadable)) modf(double x, __global double *
iptr);
double2 __attribute__((overloadable)) modf(double2 x, __global double2 *
iptr);
double3 __attribute__((overloadable)) modf(double3 x, __global double3 *
iptr);
double4 __attribute__((overloadable)) modf(double4 x, __global double4 *
iptr);
double8 __attribute__((overloadable)) modf(double8 x, __global double8 *
iptr);
double16 __attribute__((overloadable)) modf(double16 x, __global double16 *
iptr);
double __attribute__((overloadable)) modf(double x, __local double *
iptr);
double2 __attribute__((overloadable)) modf(double2 x, __local double2 *
iptr);
double3 __attribute__((overloadable)) modf(double3 x, __local double3 *
iptr);
double4 __attribute__((overloadable)) modf(double4 x, __local double4 *
iptr);
double8 __attribute__((overloadable)) modf(double8 x, __local double8 *
iptr);
double16 __attribute__((overloadable)) modf(double16 x, __local double16 *
iptr);
double __attribute__((overloadable)) modf(double x, __private double *
iptr);
double2 __attribute__((overloadable)) modf(double2 x, __private double2 *
iptr);
double3 __attribute__((overloadable)) modf(double3 x, __private double3 *
iptr);
double4 __attribute__((overloadable)) modf(double4 x, __private double4 *
iptr);
double8 __attribute__((overloadable)) modf(double8 x, __private double8 *
iptr);
double16 __attribute__((overloadable)) modf(double16 x, __private double16 *
iptr);
//half __attribute__((overloadable)) modf(half x, __global half *iptr);
//half2 __attribute__((overloadable)) modf(half2 x, __global half2 *iptr);
//half3 __attribute__((overloadable)) modf(half3 x, __global half3 *iptr);
//half4 __attribute__((overloadable)) modf(half4 x, __global half4 *iptr);
//half8 __attribute__((overloadable)) modf(half8 x, __global half8 *iptr);
//half16 __attribute__((overloadable)) modf(half16 x, __global half16 *iptr);
//half __attribute__((overloadable)) modf(half x, __local half *iptr);
//half2 __attribute__((overloadable)) modf(half2 x, __local half2 *iptr);
//half3 __attribute__((overloadable)) modf(half3 x, __local half3 *iptr);
//half4 __attribute__((overloadable)) modf(half4 x, __local half4 *iptr);
//half8 __attribute__((overloadable)) modf(half8 x, __local half8 *iptr);
//half16 __attribute__((overloadable)) modf(half16 x, __local half16 *iptr);
//half __attribute__((overloadable)) modf(half x, __private half *iptr);
//half2 __attribute__((overloadable)) modf(half2 x, __private half2 *iptr);
//half3 __attribute__((overloadable)) modf(half3 x, __private half3 *iptr);
//half4 __attribute__((overloadable)) modf(half4 x, __private half4 *iptr);
//half8 __attribute__((overloadable)) modf(half8 x, __private half8 *iptr);
//half16 __attribute__((overloadable)) modf(half16 x, __private half16 *iptr);

/**
 * Returns a quiet NaN. The nancode may be placed
 * in the significand of the resulting NaN.
 */
float const_func __attribute__((overloadable)) nan(uint nancode);
float2 const_func __attribute__((overloadable)) nan(uint2 nancode);
float3 const_func __attribute__((overloadable)) nan(uint3 nancode);
```

```
float4 const_func __attribute__((overloadable)) nan(uint4 nancode);
float8 const_func __attribute__((overloadable)) nan(uint8 nancode);
float16 const_func __attribute__((overloadable)) nan(uint16 nancode);
double const_func __attribute__((overloadable)) nan(ulong nancode);
double2 const_func __attribute__((overloadable)) nan(ulong2 nancode);
double3 const_func __attribute__((overloadable)) nan(ulong3 nancode);
double4 const_func __attribute__((overloadable)) nan(ulong4 nancode);
double8 const_func __attribute__((overloadable)) nan(ulong8 nancode);
double16 const_func __attribute__((overloadable)) nan(ulong16 nancode);
//half const_func __attribute__((overloadable)) nan(ushort nancode);
//half2 const_func __attribute__((overloadable)) nan(ushort2 nancode);
//half3 const_func __attribute__((overloadable)) nan(ushort3 nancode);
//half4 const_func __attribute__((overloadable)) nan(ushort4 nancode);
//half8 const_func __attribute__((overloadable)) nan(ushort8 nancode);
//half16 const_func __attribute__((overloadable)) nan(ushort16 nancode);

/**
 * Computes the next representable single-precision
 * floating-point value following x in the direction of
 * y. Thus, if y is less than x, nextafter() returns the
 * largest representable floating-point number less
 * than x.
 */
float const_func __attribute__((overloadable)) nextafter(float x, float y);
float2 const_func __attribute__((overloadable)) nextafter(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) nextafter(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) nextafter(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) nextafter(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) nextafter(float16 x, float16 y);
double const_func __attribute__((overloadable)) nextafter(double x, double y);
double2 const_func __attribute__((overloadable)) nextafter(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) nextafter(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) nextafter(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) nextafter(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) nextafter(double16 x, double16 y);
//half const_func __attribute__((overloadable)) nextafter(half x, half y);
//half2 const_func __attribute__((overloadable)) nextafter(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) nextafter(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) nextafter(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) nextafter(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) nextafter(half16 x, half16 y);

/**
 * Compute x to the power y.
 */
double const_func __attribute__((overloadable)) pow(double x, double y);
double2 const_func __attribute__((overloadable)) pow(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) pow(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) pow(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) pow(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) pow(double16 x, double16 y);
float const_func __attribute__((overloadable)) pow(float x, float y);
float2 const_func __attribute__((overloadable)) pow(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) pow(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) pow(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) pow(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) pow(float16 x, float16 y);
//half const_func __attribute__((overloadable)) pow(half x, half y);
//half2 const_func __attribute__((overloadable)) pow(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) pow(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) pow(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) pow(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) pow(half16 x, half16 y);
```

```

/**
 * Compute x to the power y, where y is an integer.
 */
float const_func __attribute__((overloadable)) pown(float x, int y);
float2 const_func __attribute__((overloadable)) pown(float2 x, int2 y);
float3 const_func __attribute__((overloadable)) pown(float3 x, int3 y);
float4 const_func __attribute__((overloadable)) pown(float4 x, int4 y);
float8 const_func __attribute__((overloadable)) pown(float8 x, int8 y);
float16 const_func __attribute__((overloadable)) pown(float16 x, int16 y);
double const_func __attribute__((overloadable)) pown(double x, int y);
double2 const_func __attribute__((overloadable)) pown(double2 x, int2 y);
double3 const_func __attribute__((overloadable)) pown(double3 x, int3 y);
double4 const_func __attribute__((overloadable)) pown(double4 x, int4 y);
double8 const_func __attribute__((overloadable)) pown(double8 x, int8 y);
double16 const_func __attribute__((overloadable)) pown(double16 x, int16 y);
//half const_func __attribute__((overloadable)) pown(half x, int y);
//half2 const_func __attribute__((overloadable)) pown(half2 x, int2 y);
//half3 const_func __attribute__((overloadable)) pown(half3 x, int3 y);
//half4 const_func __attribute__((overloadable)) pown(half4 x, int4 y);
//half8 const_func __attribute__((overloadable)) pown(half8 x, int8 y);
//half16 const_func __attribute__((overloadable)) pown(half16 x, int16 y);

/**
 * Compute x to the power y, where x is >= 0.
 */
float const_func __attribute__((overloadable)) powr(float x, float y);
float2 const_func __attribute__((overloadable)) powr(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) powr(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) powr(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) powr(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) powr(float16 x, float16 y);
double const_func __attribute__((overloadable)) powr(double x, double y);
double2 const_func __attribute__((overloadable)) powr(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) powr(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) powr(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) powr(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) powr(double16 x, double16 y);
//half const_func __attribute__((overloadable)) powr(half x, half y);
//half2 const_func __attribute__((overloadable)) powr(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) powr(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) powr(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) powr(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) powr(half16 x, half16 y);

/**
 * Compute the value r such that r = x - n*y, where n
 * is the integer nearest the exact value of x/y. If there
 * are two integers closest to x/y, n shall be the even
 * one. If r is zero, it is given the same sign as x.
 */
float const_func __attribute__((overloadable)) remainder(float x, float y);
float2 const_func __attribute__((overloadable)) remainder(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) remainder(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) remainder(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) remainder(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) remainder(float16 x, float16 y);
```

```
double const_func __attribute__((overloadable)) remainder(double x, double y);
double2 const_func __attribute__((overloadable)) remainder(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) remainder(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) remainder(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) remainder(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) remainder(double16 x, double16 y);
//half const_func __attribute__((overloadable)) remainder(half x, half y);
//half2 const_func __attribute__((overloadable)) remainder(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) remainder(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) remainder(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) remainder(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) remainder(half16 x, half16 y);

/**
 * The remquo function computes the value r such
 * that r = x - n*y, where n is the integer nearest the
 * exact value of x/y. If there are two integers closest
 * to x/y, n shall be the even one. If r is zero, it is
 * given the same sign as x. This is the same value
 * that is returned by the remainder function.
 * remquo also calculates the lower seven bits of the
 * integral quotient x/y, and gives that value the same
 * sign as x/y. It stores this signed value in the object
 * pointed to by quo.
 */
float __attribute__((overloadable)) remquo(float x, float y, __global int * quo);
float2 __attribute__((overloadable)) remquo(float2 x, float2 y, __global int2 * quo);
float3 __attribute__((overloadable)) remquo(float3 x, float3 y, __global int3 * quo);
float4 __attribute__((overloadable)) remquo(float4 x, float4 y, __global int4 * quo);
float8 __attribute__((overloadable)) remquo(float8 x, float8 y, __global int8 * quo);
float16 __attribute__((overloadable)) remquo(float16 x, float16 y, __global int16 * quo);
float __attribute__((overloadable)) remquo(float x, float y, __local int * quo);
float2 __attribute__((overloadable)) remquo(float2 x, float2 y, __local int2 * quo);
float3 __attribute__((overloadable)) remquo(float3 x, float3 y, __local int3 * quo);
float4 __attribute__((overloadable)) remquo(float4 x, float4 y, __local int4 * quo);
float8 __attribute__((overloadable)) remquo(float8 x, float8 y, __local int8 * quo);
float16 __attribute__((overloadable)) remquo(float16 x, float16 y, __local int16 * quo);
float __attribute__((overloadable)) remquo(float x, float y, __private int * quo);
float2 __attribute__((overloadable)) remquo(float2 x, float2 y, __private int2 * quo);
float3 __attribute__((overloadable)) remquo(float3 x, float3 y, __private int3 * quo);
float4 __attribute__((overloadable)) remquo(float4 x, float4 y, __private int4 * quo);
float8 __attribute__((overloadable)) remquo(float8 x, float8 y, __private int8 * quo);
float16 __attribute__((overloadable)) remquo(float16 x, float16 y, __private int16 * quo);
double __attribute__((overloadable)) remquo(double x, double y, __global int * quo);
double2 __attribute__((overloadable)) remquo(double2 x, double2 y, __global int2 * quo);
double3 __attribute__((overloadable)) remquo(double3 x, double3 y, __global int3 * quo);
double4 __attribute__((overloadable)) remquo(double4 x, double4 y, __global int4 * quo);
double8 __attribute__((overloadable)) remquo(double8 x, double8 y, __global int8 * quo);
double16 __attribute__((overloadable)) remquo(double16 x, double16 y, __global int16 *quo);
double __attribute__((overloadable)) remquo(double x, double y, __local int * quo);
double2 __attribute__((overloadable)) remquo(double2 x, double2 y, __local int2 * quo);
double3 __attribute__((overloadable)) remquo(double3 x, double3 y, __local int3 * quo);
double4 __attribute__((overloadable)) remquo(double4 x, double4 y, __local int4 * quo);
```

```
double8 __attribute__((overloadable)) remquo(double8 x, double8 y, __local int8 *quo);
double16 __attribute__((overloadable)) remquo(double16 x, double16 y, __local int16 *quo);
double __attribute__((overloadable)) remquo(double x, double y, __private int *quo);
double2 __attribute__((overloadable)) remquo(double2 x, double2 y, __private int2 *quo);
double3 __attribute__((overloadable)) remquo(double3 x, double3 y, __private int3 *quo);
double4 __attribute__((overloadable)) remquo(double4 x, double4 y, __private int4 *quo);
double8 __attribute__((overloadable)) remquo(double8 x, double8 y, __private int8 *quo);
double16 __attribute__((overloadable)) remquo(double16 x, double16 y, __private int16 *quo);
//half __attribute__((overloadable)) remquo(half x, half y, __global int *quo);
//half2 __attribute__((overloadable)) remquo(half2 x, half2 y, __global int2 *quo);
//half3 __attribute__((overloadable)) remquo(half3 x, half3 y, __global int3 *quo);
//half4 __attribute__((overloadable)) remquo(half4 x, half4 y, __global int4 *quo);
//half8 __attribute__((overloadable)) remquo(half8 x, half8 y, __global int8 *quo);
//half16 __attribute__((overloadable)) remquo(half16 x, half16 y, __global int16 *quo);
//half __attribute__((overloadable)) remquo(half x, half y, __local int *quo);
//half2 __attribute__((overloadable)) remquo(half2 x, half2 y, __local int2 *quo);
//half3 __attribute__((overloadable)) remquo(half3 x, half3 y, __local int3 *quo);
//half4 __attribute__((overloadable)) remquo(half4 x, half4 y, __local int4 *quo);
//half8 __attribute__((overloadable)) remquo(half8 x, half8 y, __local int8 *quo);
//half16 __attribute__((overloadable)) remquo(half16 x, half16 y, __local int16 *quo);
//half __attribute__((overloadable)) remquo(half x, half y, __private int *quo);
//half2 __attribute__((overloadable)) remquo(half2 x, half2 y, __private int2 *quo);
//half3 __attribute__((overloadable)) remquo(half3 x, half3 y, __private int3 *quo);
//half4 __attribute__((overloadable)) remquo(half4 x, half4 y, __private int4 *quo);
//half8 __attribute__((overloadable)) remquo(half8 x, half8 y, __private int8 *quo);
//half16 __attribute__((overloadable)) remquo(half16 x, half16 y, __private int16 *quo);

/**
 * Round to integral value (using round to nearest
 * even rounding mode) in floating-point format.
 * Refer to section 7.1 for description of rounding
 * modes.
 */
float const_func __attribute__((overloadable)) rint(float);
float2 const_func __attribute__((overloadable)) rint(float2);
float3 const_func __attribute__((overloadable)) rint(float3);
float4 const_func __attribute__((overloadable)) rint(float4);
float8 const_func __attribute__((overloadable)) rint(float8);
float16 const_func __attribute__((overloadable)) rint(float16);
double const_func __attribute__((overloadable)) rint(double);
double2 const_func __attribute__((overloadable)) rint(double2);
double3 const_func __attribute__((overloadable)) rint(double3);
double4 const_func __attribute__((overloadable)) rint(double4);
double8 const_func __attribute__((overloadable)) rint(double8);
double16 const_func __attribute__((overloadable)) rint(double16);
//half const_func __attribute__((overloadable)) rint(half);
//half2 const_func __attribute__((overloadable)) rint(half2);
//half3 const_func __attribute__((overloadable)) rint(half3);
//half4 const_func __attribute__((overloadable)) rint(half4);
//half8 const_func __attribute__((overloadable)) rint(half8);
//half16 const_func __attribute__((overloadable)) rint(half16);

/**
 * Compute x to the power 1/y.
 */
float const_func __attribute__((overloadable)) rootn(float x, int y);
float2 const_func __attribute__((overloadable)) rootn(float2 x, int2 y);
float3 const_func __attribute__((overloadable)) rootn(float3 x, int3 y);
float4 const_func __attribute__((overloadable)) rootn(float4 x, int4 y);
float8 const_func __attribute__((overloadable)) rootn(float8 x, int8 y);
float16 const_func __attribute__((overloadable)) rootn(float16 x, int16 y);
double const_func __attribute__((overloadable)) rootn(double x, int y);
double2 const_func __attribute__((overloadable)) rootn(double2 x, int2 y);
double3 const_func __attribute__((overloadable)) rootn(double3 x, int3 y);
double4 const_func __attribute__((overloadable)) rootn(double4 x, int4 y);
double8 const_func __attribute__((overloadable)) rootn(double8 x, int8 y);
double16 const_func __attribute__((overloadable)) rootn(double16 x, int16 y);
```

```
//half const_func __attribute__((overloadable)) rootn(half x, int y);
//half2 const_func __attribute__((overloadable)) rootn(half2 x, int2 y);
//half3 const_func __attribute__((overloadable)) rootn(half3 x, int3 y);
//half4 const_func __attribute__((overloadable)) rootn(half4 x, int4 y);
//half8 const_func __attribute__((overloadable)) rootn(half8 x, int8 y);
//half16 const_func __attribute__((overloadable)) rootn(half16 x, int16 y);

/**
 * Return the integral value nearest to x rounding
 * halfway cases away from zero, regardless of the
 * current rounding direction.
 */
float const_func __attribute__((overloadable)) round(float x);
float2 const_func __attribute__((overloadable)) round(float2 x);
float3 const_func __attribute__((overloadable)) round(float3 x);
float4 const_func __attribute__((overloadable)) round(float4 x);
float8 const_func __attribute__((overloadable)) round(float8 x);
float16 const_func __attribute__((overloadable)) round(float16 x);
double const_func __attribute__((overloadable)) round(double x);
double2 const_func __attribute__((overloadable)) round(double2 x);
double3 const_func __attribute__((overloadable)) round(double3 x);
double4 const_func __attribute__((overloadable)) round(double4 x);
double8 const_func __attribute__((overloadable)) round(double8 x);
double16 const_func __attribute__((overloadable)) round(double16 x);
//half const_func __attribute__((overloadable)) round(half x);
//half2 const_func __attribute__((overloadable)) round(half2 x);
//half3 const_func __attribute__((overloadable)) round(half3 x);
//half4 const_func __attribute__((overloadable)) round(half4 x);
//half8 const_func __attribute__((overloadable)) round(half8 x);
//half16 const_func __attribute__((overloadable)) round(half16 x);

/**
 * Compute inverse square root.
 */
float const_func __attribute__((overloadable)) rsqrt(float);
float2 const_func __attribute__((overloadable)) rsqrt(float2);
float3 const_func __attribute__((overloadable)) rsqrt(float3);
float4 const_func __attribute__((overloadable)) rsqrt(float4);
float8 const_func __attribute__((overloadable)) rsqrt(float8);
float16 const_func __attribute__((overloadable)) rsqrt(float16);
double const_func __attribute__((overloadable)) rsqrt(double);
double2 const_func __attribute__((overloadable)) rsqrt(double2);
double3 const_func __attribute__((overloadable)) rsqrt(double3);
double4 const_func __attribute__((overloadable)) rsqrt(double4);
double8 const_func __attribute__((overloadable)) rsqrt(double8);
double16 const_func __attribute__((overloadable)) rsqrt(double16);
//half const_func __attribute__((overloadable)) rsqrt(half);
//half2 const_func __attribute__((overloadable)) rsqrt(half2);
//half3 const_func __attribute__((overloadable)) rsqrt(half3);
//half4 const_func __attribute__((overloadable)) rsqrt(half4);
//half8 const_func __attribute__((overloadable)) rsqrt(half8);
//half16 const_func __attribute__((overloadable)) rsqrt(half16);

/**
 * Compute sine.
 */
double const_func __attribute__((overloadable)) sin(double);
double2 const_func __attribute__((overloadable)) sin(double2);
double3 const_func __attribute__((overloadable)) sin(double3);
double4 const_func __attribute__((overloadable)) sin(double4);
double8 const_func __attribute__((overloadable)) sin(double8);
double16 const_func __attribute__((overloadable)) sin(double16);
float const_func __attribute__((overloadable)) sin(float);
float2 const_func __attribute__((overloadable)) sin(float2);
float3 const_func __attribute__((overloadable)) sin(float3);
float4 const_func __attribute__((overloadable)) sin(float4);
float8 const_func __attribute__((overloadable)) sin(float8);
float16 const_func __attribute__((overloadable)) sin(float16);
//half const_func __attribute__((overloadable)) sin(half);
//half2 const_func __attribute__((overloadable)) sin(half2);
//half3 const_func __attribute__((overloadable)) sin(half3);
//half4 const_func __attribute__((overloadable)) sin(half4);
//half8 const_func __attribute__((overloadable)) sin(half8);
//half16 const_func __attribute__((overloadable)) sin(half16);
```

```
/**
 * Compute sine and cosine of x. The computed sine
 * is the return value and computed cosine is returned
 * in cosval.
 */
float __attribute__((overloadable)) sincos(float x, __global float *cosval);
float2 __attribute__((overloadable)) sincos(float2 x, __global float2 *cosval);
float3 __attribute__((overloadable)) sincos(float3 x, __global float3 *cosval);
float4 __attribute__((overloadable)) sincos(float4 x, __global float4 *cosval);
float8 __attribute__((overloadable)) sincos(float8 x, __global float8 *cosval);
float16 __attribute__((overloadable)) sincos(float16 x, __global float16 *cosval);
float __attribute__((overloadable)) sincos(float x, __local float *cosval);
float2 __attribute__((overloadable)) sincos(float2 x, __local float2 *cosval);
float3 __attribute__((overloadable)) sincos(float3 x, __local float3 *cosval);
float4 __attribute__((overloadable)) sincos(float4 x, __local float4 *cosval);
float8 __attribute__((overloadable)) sincos(float8 x, __local float8 *cosval);
float16 __attribute__((overloadable)) sincos(float16 x, __local float16 *cosval);
float __attribute__((overloadable)) sincos(float x, __private float *cosval);
float2 __attribute__((overloadable)) sincos(float2 x, __private float2 *cosval);
float3 __attribute__((overloadable)) sincos(float3 x, __private float3 *cosval);
float4 __attribute__((overloadable)) sincos(float4 x, __private float4 *cosval);
float8 __attribute__((overloadable)) sincos(float8 x, __private float8 *cosval);
float16 __attribute__((overloadable)) sincos(float16 x, __private float16 *cosval);
double __attribute__((overloadable)) sincos(double x, __global double *cosval);
double2 __attribute__((overloadable)) sincos(double2 x, __global double2 *cosval);
double3 __attribute__((overloadable)) sincos(double3 x, __global double3 *cosval);
double4 __attribute__((overloadable)) sincos(double4 x, __global double4 *cosval);
double8 __attribute__((overloadable)) sincos(double8 x, __global double8 *cosval);
double16 __attribute__((overloadable)) sincos(double16 x, __global double16 *cosval);
double __attribute__((overloadable)) sincos(double x, __local double *cosval);
double2 __attribute__((overloadable)) sincos(double2 x, __local double2 *cosval);
double3 __attribute__((overloadable)) sincos(double3 x, __local double3 *cosval);
double4 __attribute__((overloadable)) sincos(double4 x, __local double4 *cosval);
double8 __attribute__((overloadable)) sincos(double8 x, __local double8 *cosval);
double16 __attribute__((overloadable)) sincos(double16 x, __local double16 *cosval);
double __attribute__((overloadable)) sincos(double x, __private double *cosval);
double2 __attribute__((overloadable)) sincos(double2 x, __private double2 *cosval);
double3 __attribute__((overloadable)) sincos(double3 x, __private double3 *cosval);
double4 __attribute__((overloadable)) sincos(double4 x, __private double4 *cosval);
double8 __attribute__((overloadable)) sincos(double8 x, __private double8 *cosval);
double16 __attribute__((overloadable)) sincos(double16 x, __private double16 *cosval);
//half __attribute__((overloadable)) sincos(half x, __global half *cosval);
//half2 __attribute__((overloadable)) sincos(half2 x, __global half2 *cosval);
//half3 __attribute__((overloadable)) sincos(half3 x, __global half3 *cosval);
//half4 __attribute__((overloadable)) sincos(half4 x, __global half4 *cosval);
//half8 __attribute__((overloadable)) sincos(half8 x, __global half8 *cosval);
//half16 __attribute__((overloadable)) sincos(half16 x, __global half16 *cosval);
//half __attribute__((overloadable)) sincos(half x, __local half *cosval);
//half2 __attribute__((overloadable)) sincos(half2 x, __local half2 *cosval);
//half3 __attribute__((overloadable)) sincos(half3 x, __local half3 *cosval);
//half4 __attribute__((overloadable)) sincos(half4 x, __local half4 *cosval);
```

```
//half8 __attribute__((overloadable)) sincos(half8 x, __local half8 *cosval);
//half16 __attribute__((overloadable)) sincos(half16 x, __local half16 *cosval);
//half __attribute__((overloadable)) sincos(half x, __private half *cosval);
//half2 __attribute__((overloadable)) sincos(half2 x, __private half2 *cosval);
//half3 __attribute__((overloadable)) sincos(half3 x, __private half3 *cosval);
//half4 __attribute__((overloadable)) sincos(half4 x, __private half4 *cosval);
//half8 __attribute__((overloadable)) sincos(half8 x, __private half8 *cosval);
//half16 __attribute__((overloadable)) sincos(half16 x, __private half16 *cosval);

/**
 * Compute hyperbolic sine.
 */
float const_func __attribute__((overloadable)) sinh(float);
float2 const_func __attribute__((overloadable)) sinh(float2);
float3 const_func __attribute__((overloadable)) sinh(float3);
float4 const_func __attribute__((overloadable)) sinh(float4);
float8 const_func __attribute__((overloadable)) sinh(float8);
float16 const_func __attribute__((overloadable)) sinh(float16);
double const_func __attribute__((overloadable)) sinh(double);
double2 const_func __attribute__((overloadable)) sinh(double2);
double3 const_func __attribute__((overloadable)) sinh(double3);
double4 const_func __attribute__((overloadable)) sinh(double4);
double8 const_func __attribute__((overloadable)) sinh(double8);
double16 const_func __attribute__((overloadable)) sinh(double16);
//half const_func __attribute__((overloadable)) sinh(half);
//half2 const_func __attribute__((overloadable)) sinh(half2);
//half3 const_func __attribute__((overloadable)) sinh(half3);
//half4 const_func __attribute__((overloadable)) sinh(half4);
//half8 const_func __attribute__((overloadable)) sinh(half8);
//half16 const_func __attribute__((overloadable)) sinh(half16);

/**
 * Compute sin (PI * x).
 */
float const_func __attribute__((overloadable)) sinpi(float
x);
float2 const_func __attribute__((overloadable)) sinpi(float2
x);
float3 const_func __attribute__((overloadable)) sinpi(float3
x);
float4 const_func __attribute__((overloadable)) sinpi(float4
x);
float8 const_func __attribute__((overloadable)) sinpi(float8
x);
float16 const_func __attribute__((overloadable)) sinpi(float16
x);
double const_func __attribute__((overloadable)) sinpi(double
x);
double2 const_func __attribute__((overloadable)) sinpi(double2
x);
double3 const_func __attribute__((overloadable)) sinpi(double3
x);
double4 const_func __attribute__((overloadable)) sinpi(double4
x);
double8 const_func __attribute__((overloadable)) sinpi(double8
x);
double16 const_func __attribute__((overloadable)) sinpi(double16
x);
//half const_func __attribute__((overloadable)) sinpi(half x);
//half2 const_func __attribute__((overloadable)) sinpi(half2 x);
//half3 const_func __attribute__((overloadable)) sinpi(half3 x);
//half4 const_func __attribute__((overloadable)) sinpi(half4 x);
//half8 const_func __attribute__((overloadable)) sinpi(half8 x);
//half16 const_func __attribute__((overloadable)) sinpi(half16 x);

/**
 * Compute square root.
 */
double const_func __attribute__((overloadable)) sqrt(double);
double2 const_func __attribute__((overloadable)) sqrt(double2);
double3 const_func __attribute__((overloadable)) sqrt(double3);
double4 const_func __attribute__((overloadable)) sqrt(double4);
double8 const_func __attribute__((overloadable)) sqrt(double8);
double16 const_func __attribute__((overloadable)) sqrt(double16);
float const_func __attribute__((overloadable)) sqrt(float);
float2 const_func __attribute__((overloadable)) sqrt(float2);
float3 const_func __attribute__((overloadable)) sqrt(float3);
float4 const_func __attribute__((overloadable)) sqrt(float4);
float8 const_func __attribute__((overloadable)) sqrt(float8);
float16 const_func __attribute__((overloadable)) sqrt(float16);
//half const_func __attribute__((overloadable)) sqrt(half);
//half2 const_func __attribute__((overloadable)) sqrt(half2);
//half3 const_func __attribute__((overloadable)) sqrt(half3);
//half4 const_func __attribute__((overloadable)) sqrt(half4);
//half8 const_func __attribute__((overloadable)) sqrt(half8);
//half16 const_func __attribute__((overloadable)) sqrt(half16);
```

```
/**
 * Compute tangent.
 */
float const_func __attribute__((overloadable)) tan(float);
float2 const_func __attribute__((overloadable)) tan(float2);
float3 const_func __attribute__((overloadable)) tan(float3);
float4 const_func __attribute__((overloadable)) tan(float4);
float8 const_func __attribute__((overloadable)) tan(float8);
float16 const_func __attribute__((overloadable)) tan(float16);
double const_func __attribute__((overloadable)) tan(double);
double2 const_func __attribute__((overloadable)) tan(double2);
double3 const_func __attribute__((overloadable)) tan(double3);
double4 const_func __attribute__((overloadable)) tan(double4);
double8 const_func __attribute__((overloadable)) tan(double8);
double16 const_func __attribute__((overloadable)) tan(double16);
//half const_func __attribute__((overloadable)) tan(half);
//half2 const_func __attribute__((overloadable)) tan(half2);
//half3 const_func __attribute__((overloadable)) tan(half3);
//half4 const_func __attribute__((overloadable)) tan(half4);
//half8 const_func __attribute__((overloadable)) tan(half8);
//half16 const_func __attribute__((overloadable)) tan(half16);

/**
 * Compute hyperbolic tangent.
 */
float const_func __attribute__((overloadable)) tanh(float);
float2 const_func __attribute__((overloadable)) tanh(float2);
float3 const_func __attribute__((overloadable)) tanh(float3);
float4 const_func __attribute__((overloadable)) tanh(float4);
float8 const_func __attribute__((overloadable)) tanh(float8);
float16 const_func __attribute__((overloadable)) tanh(float16);
double const_func __attribute__((overloadable)) tanh(double);
double2 const_func __attribute__((overloadable)) tanh(double2);
double3 const_func __attribute__((overloadable)) tanh(double3);
double4 const_func __attribute__((overloadable)) tanh(double4);
double8 const_func __attribute__((overloadable)) tanh(double8);
double16 const_func __attribute__((overloadable)) tanh(double16);
//half const_func __attribute__((overloadable)) tanh(half);
//half2 const_func __attribute__((overloadable)) tanh(half2);
//half3 const_func __attribute__((overloadable)) tanh(half3);
//half4 const_func __attribute__((overloadable)) tanh(half4);
//half8 const_func __attribute__((overloadable)) tanh(half8);
//half16 const_func __attribute__((overloadable)) tanh(half16);

/**
 * Compute tan (PI * x).
 */
float const_func __attribute__((overloadable)) tanpi(float x);
float2 const_func __attribute__((overloadable)) tanpi(float2 x);
float3 const_func __attribute__((overloadable)) tanpi(float3 x);
float4 const_func __attribute__((overloadable)) tanpi(float4 x);
float8 const_func __attribute__((overloadable)) tanpi(float8 x);
float16 const_func __attribute__((overloadable)) tanpi(float16 x);
double const_func __attribute__((overloadable)) tanpi(double x);
double2 const_func __attribute__((overloadable)) tanpi(double2 x);
double3 const_func __attribute__((overloadable)) tanpi(double3 x);
double4 const_func __attribute__((overloadable)) tanpi(double4 x);
double8 const_func __attribute__((overloadable)) tanpi(double8 x);
double16 const_func __attribute__((overloadable)) tanpi(double16 x);
//half const_func __attribute__((overloadable)) tanpi(half x);
//half2 const_func __attribute__((overloadable)) tanpi(half2 x);
//half3 const_func __attribute__((overloadable)) tanpi(half3 x);
//half4 const_func __attribute__((overloadable)) tanpi(half4 x);
//half8 const_func __attribute__((overloadable)) tanpi(half8 x);
//half16 const_func __attribute__((overloadable)) tanpi(half16 x);

/**
 * Compute the gamma function.
 */
float const_func __attribute__((overloadable)) tgamma(float);
float2 const_func __attribute__((overloadable)) tgamma(float2);
float3 const_func __attribute__((overloadable)) tgamma(float3);
float4 const_func __attribute__((overloadable)) tgamma(float4);
float8 const_func __attribute__((overloadable)) tgamma(float8);
float16 const_func __attribute__((overloadable)) tgamma(float16);
```

```
double const_func __attribute__((overloadable)) tgamma(double);
double2 const_func __attribute__((overloadable)) tgamma(double2);
double3 const_func __attribute__((overloadable)) tgamma(double3);
double4 const_func __attribute__((overloadable)) tgamma(double4);
double8 const_func __attribute__((overloadable)) tgamma(double8);
double16 const_func __attribute__((overloadable)) tgamma(double16);
//half const_func __attribute__((overloadable)) tgamma(half);
//half2 const_func __attribute__((overloadable)) tgamma(half2);
//half3 const_func __attribute__((overloadable)) tgamma(half3);
//half4 const_func __attribute__((overloadable)) tgamma(half4);
//half8 const_func __attribute__((overloadable)) tgamma(half8);
//half16 const_func __attribute__((overloadable)) tgamma(half16);

/**
 * Round to integral value using the round to zero
 * rounding mode.
 */
float const_func __attribute__((overloadable)) trunc(float);
float2 const_func __attribute__((overloadable)) trunc(float2);
float3 const_func __attribute__((overloadable)) trunc(float3);
float4 const_func __attribute__((overloadable)) trunc(float4);
float8 const_func __attribute__((overloadable)) trunc(float8);
float16 const_func __attribute__((overloadable)) trunc(float16);
double const_func __attribute__((overloadable)) trunc(double);
double2 const_func __attribute__((overloadable)) trunc(double2);
double3 const_func __attribute__((overloadable)) trunc(double3);
double4 const_func __attribute__((overloadable)) trunc(double4);
double8 const_func __attribute__((overloadable)) trunc(double8);
double16 const_func __attribute__((overloadable)) trunc(double16);
//half const_func __attribute__((overloadable)) trunc(half);
//half2 const_func __attribute__((overloadable)) trunc(half2);
//half3 const_func __attribute__((overloadable)) trunc(half3);
//half4 const_func __attribute__((overloadable)) trunc(half4);
//half8 const_func __attribute__((overloadable)) trunc(half8);
//half16 const_func __attribute__((overloadable)) trunc(half16);

/**
 * Compute cosine. x must be in the range -2^16 ... +2^16.
 */
float const_func __attribute__((overloadable)) half_cos(float
x);
float2 const_func __attribute__((overloadable)) half_cos(float2
x);
float3 const_func __attribute__((overloadable)) half_cos(float3
x);
float4 const_func __attribute__((overloadable)) half_cos(float4
x);
float8 const_func __attribute__((overloadable)) half_cos(float8
x);
float16 const_func __attribute__((overloadable)) half_cos(float16
x);

/**
 * Compute x / y.
 */
float const_func __attribute__((overloadable)) half_divide(float
x, float y);
float2 const_func __attribute__((overloadable)) half_divide(float2
x, float2 y);
float3 const_func __attribute__((overloadable)) half_divide(float3
x, float3 y);
float4 const_func __attribute__((overloadable)) half_divide(float4
x, float4 y);
float8 const_func __attribute__((overloadable)) half_divide(float8
x, float8 y);
float16 const_func __attribute__((overloadable)) half_divide(float16
x, float16 y);

/**
 * Compute the base- e exponential of x.
 */
float const_func __attribute__((overloadable)) half_exp(float
x);
float2 const_func __attribute__((overloadable)) half_exp(float2
x);
float3 const_func __attribute__((overloadable)) half_exp(float3
x);
float4 const_func __attribute__((overloadable)) half_exp(float4
x);
float8 const_func __attribute__((overloadable)) half_exp(float8
x);
float16 const_func __attribute__((overloadable)) half_exp(float16
x);

/**
 * Compute the base- 2 exponential of x.
 */
```

```
float const_func __attribute__((overloadable)) half_exp2(float x);
float2 const_func __attribute__((overloadable)) half_exp2(float2 x);
float3 const_func __attribute__((overloadable)) half_exp2(float3 x);
float4 const_func __attribute__((overloadable)) half_exp2(float4 x);
float8 const_func __attribute__((overloadable)) half_exp2(float8 x);
float16 const_func __attribute__((overloadable)) half_exp2(float16 x);

/**
 * Compute the base- 10 exponential of x.
 */
float const_func __attribute__((overloadable)) half_exp10(float x);
float2 const_func __attribute__((overloadable)) half_exp10(float2 x);
float3 const_func __attribute__((overloadable)) half_exp10(float3 x);
float4 const_func __attribute__((overloadable)) half_exp10(float4 x);
float8 const_func __attribute__((overloadable)) half_exp10(float8 x);
float16 const_func __attribute__((overloadable)) half_exp10(float16 x);

/**
 * Compute natural logarithm.
 */
float const_func __attribute__((overloadable)) half_log(float x);
float2 const_func __attribute__((overloadable)) half_log(float2 x);
float3 const_func __attribute__((overloadable)) half_log(float3 x);
float4 const_func __attribute__((overloadable)) half_log(float4 x);
float8 const_func __attribute__((overloadable)) half_log(float8 x);
float16 const_func __attribute__((overloadable)) half_log(float16 x);

/**
 * Compute a base 2 logarithm.
 */
float const_func __attribute__((overloadable)) half_log2(float x);
float2 const_func __attribute__((overloadable)) half_log2(float2 x);
float3 const_func __attribute__((overloadable)) half_log2(float3 x);
float4 const_func __attribute__((overloadable)) half_log2(float4 x);
float8 const_func __attribute__((overloadable)) half_log2(float8 x);
float16 const_func __attribute__((overloadable)) half_log2(float16 x);

/**
 * Compute a base 10 logarithm.
 */
float const_func __attribute__((overloadable)) half_log10(float x);
float2 const_func __attribute__((overloadable)) half_log10(float2 x);
float3 const_func __attribute__((overloadable)) half_log10(float3 x);
float4 const_func __attribute__((overloadable)) half_log10(float4 x);
float8 const_func __attribute__((overloadable)) half_log10(float8 x);
float16 const_func __attribute__((overloadable)) half_log10(float16 x);

/**
 * Compute x to the power y, where x is >= 0.
 */
float const_func __attribute__((overloadable)) half_powr(float x, float y);
float2 const_func __attribute__((overloadable)) half_powr(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) half_powr(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) half_powr(float4
```

```
x, float4 y);
float8 const_func __attribute__((overloadable)) half_powr(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) half_powr(float16 x, float16 y);

/**
 * Compute reciprocal.
 */
float const_func __attribute__((overloadable)) half_recip(float x);
float2 const_func __attribute__((overloadable)) half_recip(float2 x);
float3 const_func __attribute__((overloadable)) half_recip(float3 x);
float4 const_func __attribute__((overloadable)) half_recip(float4 x);
float8 const_func __attribute__((overloadable)) half_recip(float8 x);
float16 const_func __attribute__((overloadable)) half_recip(float16 x);

/**
 * Compute inverse square root.
 */
float const_func __attribute__((overloadable)) half_rsqrt(float x);
float2 const_func __attribute__((overloadable)) half_rsqrt(float2 x);
float3 const_func __attribute__((overloadable)) half_rsqrt(float3 x);
float4 const_func __attribute__((overloadable)) half_rsqrt(float4 x);
float8 const_func __attribute__((overloadable)) half_rsqrt(float8 x);
float16 const_func __attribute__((overloadable)) half_rsqrt(float16 x);

/**
 * Compute sine. x must be in the range -2^16 ... +2^16.
 */
float const_func __attribute__((overloadable)) half_sin(float x);
float2 const_func __attribute__((overloadable)) half_sin(float2 x);
float3 const_func __attribute__((overloadable)) half_sin(float3 x);
float4 const_func __attribute__((overloadable)) half_sin(float4 x);
float8 const_func __attribute__((overloadable)) half_sin(float8 x);
float16 const_func __attribute__((overloadable)) half_sin(float16 x);

/**
 * Compute square root.
 */
float const_func __attribute__((overloadable)) half_sqrt(float x);
float2 const_func __attribute__((overloadable)) half_sqrt(float2 x);
float3 const_func __attribute__((overloadable)) half_sqrt(float3 x);
float4 const_func __attribute__((overloadable)) half_sqrt(float4 x);
float8 const_func __attribute__((overloadable)) half_sqrt(float8 x);
float16 const_func __attribute__((overloadable)) half_sqrt(float16 x);

/**
 * Compute tangent. x must be in the range -216 ... +216.
 */
float const_func __attribute__((overloadable)) half_tan(float x);
float2 const_func __attribute__((overloadable)) half_tan(float2 x);
float3 const_func __attribute__((overloadable)) half_tan(float3 x);
float4 const_func __attribute__((overloadable)) half_tan(float4 x);
float8 const_func __attribute__((overloadable)) half_tan(float8 x);
float16 const_func __attribute__((overloadable)) half_tan(float16 x);

/**
```

```
 * Compute cosine over an implementation-defined range.
 * The maximum error is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_cos(float x);
float2 const_func __attribute__((overloadable)) native_cos(float2 x);
float3 const_func __attribute__((overloadable)) native_cos(float3 x);
float4 const_func __attribute__((overloadable)) native_cos(float4 x);
float8 const_func __attribute__((overloadable)) native_cos(float8 x);
float16 const_func __attribute__((overloadable)) native_cos(float16 x);

// EXTENSION: native double
double const_func __attribute__((overloadable)) native_cos(double x);
double2 const_func __attribute__((overloadable)) native_cos(double2 x);
double3 const_func __attribute__((overloadable)) native_cos(double3 x);
double4 const_func __attribute__((overloadable)) native_cos(double4 x);
double8 const_func __attribute__((overloadable)) native_cos(double8 x);
double16 const_func __attribute__((overloadable)) native_cos(double16 x);

/**
 * Compute x / y over an implementation-defined range.
 * The maximum error is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_divide(float x, float y);
float2 const_func __attribute__((overloadable)) native_divide(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) native_divide(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) native_divide(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) native_divide(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) native_divide(float16 x, float16 y);

// EXTENSION: native double
double const_func __attribute__((overloadable)) native_divide(double x, double y);
double2 const_func __attribute__((overloadable)) native_divide(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) native_divide(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) native_divide(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) native_divide(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) native_divide(double16 x, double16 y);

/**
 * Compute the base- e exponential of x over an
 * implementation-defined range. The maximum error is
 * implementation-defined.
 */
float const_func __attribute__((overloadable)) native_exp(float x);
float2 const_func __attribute__((overloadable)) native_exp(float2 x);
float3 const_func __attribute__((overloadable)) native_exp(float3 x);
float4 const_func __attribute__((overloadable)) native_exp(float4 x);
float8 const_func __attribute__((overloadable)) native_exp(float8 x);
float16 const_func __attribute__((overloadable)) native_exp(float16 x);

/**
 * Compute the base- 2 exponential of x over an
 * implementation-defined range. The maximum error is
 * implementation-defined.
 */
float const_func __attribute__((overloadable)) native_exp2(float x);
float2 const_func __attribute__((overloadable)) native_exp2(float2
```

```
x);
float3 const_func __attribute__((overloadable)) native_exp2(float3
x);
float4 const_func __attribute__((overloadable)) native_exp2(float4
x);
float8 const_func __attribute__((overloadable)) native_exp2(float8
x);
float16 const_func __attribute__((overloadable)) native_exp2(float16
x);

/**
 * Compute the base- 10 exponential of x over an
 * implementation-defined range. The maximum error is
 * implementation-defined.
 */
float const_func __attribute__((overloadable)) native_exp10(float
x);
float2 const_func __attribute__((overloadable)) native_exp10(float2
x);
float3 const_func __attribute__((overloadable)) native_exp10(float3
x);
float4 const_func __attribute__((overloadable)) native_exp10(float4
x);
float8 const_func __attribute__((overloadable)) native_exp10(float8
x);
float16 const_func __attribute__((overloadable)) native_exp10(float16
x);

/**
 * Compute natural logarithm over an implementationdefined
 * range. The maximum error is implementation
 * defined.
 */
float const_func __attribute__((overloadable)) native_log(float
x);
float2 const_func __attribute__((overloadable)) native_log(float2
x);
float3 const_func __attribute__((overloadable)) native_log(float3
x);
float4 const_func __attribute__((overloadable)) native_log(float4
x);
float8 const_func __attribute__((overloadable)) native_log(float8
x);
float16 const_func __attribute__((overloadable)) native_log(float16
x);

/**
 * Compute a base 2 logarithm over an implementationdefined
 * range. The maximum error is implementationdefined.
 */
float const_func __attribute__((overloadable)) native_log2(float
x);
float2 const_func __attribute__((overloadable)) native_log2(float2
x);
float3 const_func __attribute__((overloadable)) native_log2(float3
x);
float4 const_func __attribute__((overloadable)) native_log2(float4
x);
float8 const_func __attribute__((overloadable)) native_log2(float8
x);
float16 const_func __attribute__((overloadable)) native_log2(float16
x);

/**
 * Compute a base 10 logarithm over an implementationdefined
 * range. The maximum error is implementationdefined.
 */
float const_func __attribute__((overloadable)) native_log10(float
x);
float2 const_func __attribute__((overloadable)) native_log10(float2
x);
float3 const_func __attribute__((overloadable)) native_log10(float3
x);
float4 const_func __attribute__((overloadable)) native_log10(float4
x);
float8 const_func __attribute__((overloadable)) native_log10(float8
x);
float16 const_func __attribute__((overloadable)) native_log10(float16
x);

/**
 * Compute x to the power y, where x is >= 0. The range of
 * x and y are implementation-defined. The maximum error
 * is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_powr(float
x, float y);
```

```
float2 const_func __attribute__((overloadable)) native_powr(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) native_powr(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) native_powr(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) native_powr(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) native_powr(float16 x, float16 y);

/**
 * Compute reciprocal over an implementation-defined
 * range. The maximum error is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_recip(float x);
float2 const_func __attribute__((overloadable)) native_recip(float2 x);
float3 const_func __attribute__((overloadable)) native_recip(float3 x);
float4 const_func __attribute__((overloadable)) native_recip(float4 x);
float8 const_func __attribute__((overloadable)) native_recip(float8 x);
float16 const_func __attribute__((overloadable)) native_recip(float16 x);

/**
 * Compute inverse square root over an implementationdefined
 * range. The maximum error is implementationdefined.
 */
float const_func __attribute__((overloadable)) native_rsqrt(float x);
float2 const_func __attribute__((overloadable)) native_rsqrt(float2 x);
float3 const_func __attribute__((overloadable)) native_rsqrt(float3 x);
float4 const_func __attribute__((overloadable)) native_rsqrt(float4 x);
float8 const_func __attribute__((overloadable)) native_rsqrt(float8 x);
float16 const_func __attribute__((overloadable)) native_rsqrt(float16 x);

/**
 * Compute sine over an implementation-defined range.
 * The maximum error is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_sin(float x);
float2 const_func __attribute__((overloadable)) native_sin(float2 x);
float3 const_func __attribute__((overloadable)) native_sin(float3 x);
float4 const_func __attribute__((overloadable)) native_sin(float4 x);
float8 const_func __attribute__((overloadable)) native_sin(float8 x);
float16 const_func __attribute__((overloadable)) native_sin(float16 x);

/**
 * Compute square root over an implementation-defined
 * range. The maximum error is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_sqrt(float x);
float2 const_func __attribute__((overloadable)) native_sqrt(float2 x);
float3 const_func __attribute__((overloadable)) native_sqrt(float3 x);
float4 const_func __attribute__((overloadable)) native_sqrt(float4 x);
float8 const_func __attribute__((overloadable)) native_sqrt(float8 x);
float16 const_func __attribute__((overloadable)) native_sqrt(float16 x);

/**
 * Compute tangent over an implementation-defined range.
 * The maximum error is implementation-defined.
 */
float const_func __attribute__((overloadable)) native_tan(float x);
float2 const_func __attribute__((overloadable)) native_tan(float2 x);
```

```
float3 const_func __attribute__((overloadable)) native_tan(float3 x);
float4 const_func __attribute__((overloadable)) native_tan(float4 x);
float8 const_func __attribute__((overloadable)) native_tan(float8 x);
float16 const_func __attribute__((overloadable)) native_tan(float16 x);

// Integer functions:

/**
 * Returns | x |.
 */
uchar const_func __attribute__((overloadable)) abs(char x);
uchar const_func __attribute__((overloadable)) abs(uchar x);
uchar2 const_func __attribute__((overloadable)) abs(char2 x);
uchar2 const_func __attribute__((overloadable)) abs(uchar2 x);
uchar3 const_func __attribute__((overloadable)) abs(char3 x);
uchar3 const_func __attribute__((overloadable)) abs(uchar3 x);
uchar4 const_func __attribute__((overloadable)) abs(char4 x);
uchar4 const_func __attribute__((overloadable)) abs(uchar4 x);
uchar8 const_func __attribute__((overloadable)) abs(char8 x);
uchar8 const_func __attribute__((overloadable)) abs(uchar8 x);
uchar16 const_func __attribute__((overloadable)) abs(char16 x);
uchar16 const_func __attribute__((overloadable)) abs(uchar16 x);
ushort const_func __attribute__((overloadable)) abs(short x);
ushort const_func __attribute__((overloadable)) abs(ushort x);
ushort2 const_func __attribute__((overloadable)) abs(short2 x);
ushort2 const_func __attribute__((overloadable)) abs(ushort2 x);
ushort3 const_func __attribute__((overloadable)) abs(short3 x);
ushort3 const_func __attribute__((overloadable)) abs(ushort3 x);
ushort4 const_func __attribute__((overloadable)) abs(short4 x);
ushort4 const_func __attribute__((overloadable)) abs(ushort4 x);
ushort8 const_func __attribute__((overloadable)) abs(short8 x);
ushort8 const_func __attribute__((overloadable)) abs(ushort8 x);
ushort16 const_func __attribute__((overloadable)) abs(short16 x);
ushort16 const_func __attribute__((overloadable)) abs(ushort16 x);
uint const_func __attribute__((overloadable)) abs(int x);
uint const_func __attribute__((overloadable)) abs(uint x);
uint2 const_func __attribute__((overloadable)) abs(int2 x);
uint2 const_func __attribute__((overloadable)) abs(uint2 x);
uint3 const_func __attribute__((overloadable)) abs(int3 x);
uint3 const_func __attribute__((overloadable)) abs(uint3 x);
uint4 const_func __attribute__((overloadable)) abs(int4 x);
uint4 const_func __attribute__((overloadable)) abs(uint4 x);
uint8 const_func __attribute__((overloadable)) abs(int8 x);
uint8 const_func __attribute__((overloadable)) abs(uint8 x);
uint16 const_func __attribute__((overloadable)) abs(int16 x);
uint16 const_func __attribute__((overloadable)) abs(uint16 x);
ulong const_func __attribute__((overloadable)) abs(long x);
ulong const_func __attribute__((overloadable)) abs(ulong x);
ulong2 const_func __attribute__((overloadable)) abs(long2
```

```
x);
ulong2 const_func __attribute__((overloadable)) abs(ulong2 x);
ulong3 const_func __attribute__((overloadable)) abs(long3 x);
ulong3 const_func __attribute__((overloadable)) abs(ulong3 x);
ulong4 const_func __attribute__((overloadable)) abs(long4 x);
ulong4 const_func __attribute__((overloadable)) abs(ulong4 x);
ulong8 const_func __attribute__((overloadable)) abs(long8 x);
ulong8 const_func __attribute__((overloadable)) abs(ulong8 x);
ulong16 const_func __attribute__((overloadable)) abs(long16 x);
ulong16 const_func __attribute__((overloadable)) abs(ulong16 x);

/**
 * Returns | x - y | without modulo overflow.
 */
uchar const_func __attribute__((overloadable)) abs_diff(char x, char y);
uchar const_func __attribute__((overloadable)) abs_diff(uchar x, uchar y);
uchar2 const_func __attribute__((overloadable)) abs_diff(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) abs_diff(uchar2 x, uchar2 y);
uchar3 const_func __attribute__((overloadable)) abs_diff(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) abs_diff(uchar3 x, uchar3 y);
uchar4 const_func __attribute__((overloadable)) abs_diff(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) abs_diff(uchar4 x, uchar4 y);
uchar8 const_func __attribute__((overloadable)) abs_diff(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) abs_diff(uchar8 x, uchar8 y);
uchar16 const_func __attribute__((overloadable)) abs_diff(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) abs_diff(uchar16 x, uchar16 y);
ushort const_func __attribute__((overloadable)) abs_diff(short x, short y);
ushort const_func __attribute__((overloadable)) abs_diff(ushort x, ushort y);
ushort2 const_func __attribute__((overloadable)) abs_diff(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) abs_diff(ushort2 x, ushort2 y);
ushort3 const_func __attribute__((overloadable)) abs_diff(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) abs_diff(ushort3 x, ushort3 y);
ushort4 const_func __attribute__((overloadable)) abs_diff(short4 x, short4 y);
ushort4 const_func __attribute__((overloadable)) abs_diff(ushort4 x, ushort4 y);
ushort8 const_func __attribute__((overloadable)) abs_diff(short8 x, short8 y);
ushort8 const_func __attribute__((overloadable)) abs_diff(ushort8 x, ushort8 y);
ushort16 const_func __attribute__((overloadable)) abs_diff(short16 x, short16 y);
ushort16 const_func __attribute__((overloadable)) abs_diff(ushort16 x, ushort16 y);
uint const_func __attribute__((overloadable)) abs_diff(int x, int y);
uint const_func __attribute__((overloadable)) abs_diff(uint x, uint y);
uint2 const_func __attribute__((overloadable)) abs_diff(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) abs_diff(uint2 x, uint2 y);
uint3 const_func __attribute__((overloadable)) abs_diff(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) abs_diff(uint3 x, uint3 y);
uint4 const_func __attribute__((overloadable)) abs_diff(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) abs_diff(uint4 x, uint4 y);
```

```
uint8 const_func __attribute__((overloadable)) abs_diff(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) abs_diff(uint8 x, uint8 y);
uint16 const_func __attribute__((overloadable)) abs_diff(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) abs_diff(uint16 x, uint16 y);
ulong const_func __attribute__((overloadable)) abs_diff(long x, long y);
ulong const_func __attribute__((overloadable)) abs_diff(ulong x, ulong y);
ulong2 const_func __attribute__((overloadable)) abs_diff(long2 x, long2 y);
ulong2 const_func __attribute__((overloadable)) abs_diff(ulong2 x, ulong2 y);
ulong3 const_func __attribute__((overloadable)) abs_diff(long3 x, long3 y);
ulong3 const_func __attribute__((overloadable)) abs_diff(ulong3 x, ulong3 y);
ulong4 const_func __attribute__((overloadable)) abs_diff(long4 x, long4 y);
ulong4 const_func __attribute__((overloadable)) abs_diff(ulong4 x, ulong4 y);
ulong8 const_func __attribute__((overloadable)) abs_diff(long8 x, long8 y);
ulong8 const_func __attribute__((overloadable)) abs_diff(ulong8 x, ulong8 y);
ulong16 const_func __attribute__((overloadable)) abs_diff(long16 x, long16 y);
ulong16 const_func __attribute__((overloadable)) abs_diff(ulong16 x, ulong16 y);

/**
 * Returns x + y and saturates the result.
 */
char const_func __attribute__((overloadable)) add_sat(char x, char y);
uchar const_func __attribute__((overloadable)) add_sat(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) add_sat(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) add_sat(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) add_sat(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) add_sat(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) add_sat(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) add_sat(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) add_sat(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) add_sat(uchar8 x, uchar8 y);
char16 const_func __attribute__((overloadable)) add_sat(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) add_sat(uchar16 x, uchar16 y);
short const_func __attribute__((overloadable)) add_sat(short x, short y);
ushort const_func __attribute__((overloadable)) add_sat(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) add_sat(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) add_sat(ushort2 x, ushort2 y);
short3 const_func __attribute__((overloadable)) add_sat(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) add_sat(ushort3 x, ushort3 y);
short4 const_func __attribute__((overloadable)) add_sat(short4 x, short4 y);
ushort4 const_func __attribute__((overloadable)) add_sat(ushort4 x, ushort4 y);
short8 const_func __attribute__((overloadable)) add_sat(short8 x, short8 y);
ushort8 const_func __attribute__((overloadable)) add_sat(ushort8 x, ushort8 y);
short16 const_func __attribute__((overloadable)) add_sat(short16 x, short16 y);
ushort16 const_func __attribute__((overloadable)) add_sat(ushort16 x, ushort16 y);
int const_func __attribute__((overloadable)) add_sat(int x, int y);
uint const_func __attribute__((overloadable)) add_sat(
```

```
uint x, uint y);
int2 const_func __attribute__((overloadable)) add_sat(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) add_sat(uint2 x, uint2 y);
int3 const_func __attribute__((overloadable)) add_sat(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) add_sat(uint3 x, uint3 y);
int4 const_func __attribute__((overloadable)) add_sat(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) add_sat(uint4 x, uint4 y);
int8 const_func __attribute__((overloadable)) add_sat(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) add_sat(uint8 x, uint8 y);
int16 const_func __attribute__((overloadable)) add_sat(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) add_sat(uint16 x, uint16 y);
long const_func __attribute__((overloadable)) add_sat(long x, long y);
ulong const_func __attribute__((overloadable)) add_sat(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) add_sat(long2 x, long2 y);
ulong2 const_func __attribute__((overloadable)) add_sat(ulong2 x, ulong2 y);
long3 const_func __attribute__((overloadable)) add_sat(long3 x, long3 y);
ulong3 const_func __attribute__((overloadable)) add_sat(ulong3 x, ulong3 y);
long4 const_func __attribute__((overloadable)) add_sat(long4 x, long4 y);
ulong4 const_func __attribute__((overloadable)) add_sat(ulong4 x, ulong4 y);
long8 const_func __attribute__((overloadable)) add_sat(long8 x, long8 y);
ulong8 const_func __attribute__((overloadable)) add_sat(ulong8 x, ulong8 y);
long16 const_func __attribute__((overloadable)) add_sat(long16 x, long16 y);
ulong16 const_func __attribute__((overloadable)) add_sat(ulong16 x, ulong16 y);

/**
 * Returns (x + y) >> 1. The intermediate sum does
 * not modulo overflow.
 */
char const_func __attribute__((overloadable)) hadd(char x, char y);
uchar const_func __attribute__((overloadable)) hadd(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) hadd(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) hadd(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) hadd(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) hadd(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) hadd(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) hadd(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) hadd(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) hadd(uchar8 x, uchar8 y);
char16 const_func __attribute__((overloadable)) hadd(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) hadd(uchar16 x, uchar16 y);
short const_func __attribute__((overloadable)) hadd(short x, short y);
ushort const_func __attribute__((overloadable)) hadd(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) hadd(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) hadd(ushort2 x, ushort2 y);
short3 const_func __attribute__((overloadable)) hadd(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) hadd(ushort3 x, ushort3 y);
short4 const_func __attribute__((overloadable)) hadd(short4
```

```
x, short4 y);
ushort4 const_func __attribute__((overloadable)) hadd(ushort4 x, ushort4 y);
short8 const_func __attribute__((overloadable)) hadd(short8 x, short8 y);
ushort8 const_func __attribute__((overloadable)) hadd(ushort8 x, ushort8 y);
short16 const_func __attribute__((overloadable)) hadd(short16 x, short16 y);
ushort16 const_func __attribute__((overloadable)) hadd(ushort16 x, ushort16 y);
int const_func __attribute__((overloadable)) hadd(int x, int y);
uint const_func __attribute__((overloadable)) hadd(uint x, uint y);
int2 const_func __attribute__((overloadable)) hadd(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) hadd(uint2 x, uint2 y);
int3 const_func __attribute__((overloadable)) hadd(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) hadd(uint3 x, uint3 y);
int4 const_func __attribute__((overloadable)) hadd(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) hadd(uint4 x, uint4 y);
int8 const_func __attribute__((overloadable)) hadd(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) hadd(uint8 x, uint8 y);
int16 const_func __attribute__((overloadable)) hadd(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) hadd(uint16 x, uint16 y);
long const_func __attribute__((overloadable)) hadd(long x, long y);
ulong const_func __attribute__((overloadable)) hadd(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) hadd(long2 x, long2 y);
ulong2 const_func __attribute__((overloadable)) hadd(ulong2 x, ulong2 y);
long3 const_func __attribute__((overloadable)) hadd(long3 x, long3 y);
ulong3 const_func __attribute__((overloadable)) hadd(ulong3 x, ulong3 y);
long4 const_func __attribute__((overloadable)) hadd(long4 x, long4 y);
ulong4 const_func __attribute__((overloadable)) hadd(ulong4 x, ulong4 y);
long8 const_func __attribute__((overloadable)) hadd(long8 x, long8 y);
ulong8 const_func __attribute__((overloadable)) hadd(ulong8 x, ulong8 y);
long16 const_func __attribute__((overloadable)) hadd(long16 x, long16 y);
ulong16 const_func __attribute__((overloadable)) hadd(ulong16 x, ulong16 y);

/**
 * Returns (x + y + 1) >> 1. The intermediate sum
 * does not modulo overflow.
 */
char const_func __attribute__((overloadable)) rhadd(char x, char y);
uchar const_func __attribute__((overloadable)) rhadd(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) rhadd(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) rhadd(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) rhadd(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) rhadd(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) rhadd(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) rhadd(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) rhadd(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) rhadd(uchar8 x, uchar8 y);
char16 const_func __attribute__((overloadable)) rhadd(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) rhadd(uchar16
```

```
x, uchar16 y);
short const_func __attribute__((overloadable)) rhadd(short x, short y);
ushort const_func __attribute__((overloadable)) rhadd(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) rhadd(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) rhadd(ushort2 x, ushort2 y);
short3 const_func __attribute__((overloadable)) rhadd(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) rhadd(ushort3 x, ushort3 y);
short4 const_func __attribute__((overloadable)) rhadd(short4 x, short4 y);
ushort4 const_func __attribute__((overloadable)) rhadd(ushort4 x, ushort4 y);
short8 const_func __attribute__((overloadable)) rhadd(short8 x, short8 y);
ushort8 const_func __attribute__((overloadable)) rhadd(ushort8 x, ushort8 y);
short16 const_func __attribute__((overloadable)) rhadd(short16 x, short16 y);
ushort16 const_func __attribute__((overloadable)) rhadd(ushort16 x, ushort16 y);
int const_func __attribute__((overloadable)) rhadd(int x, int y);
uint const_func __attribute__((overloadable)) rhadd(uint x, uint y);
int2 const_func __attribute__((overloadable)) rhadd(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) rhadd(uint2 x, uint2 y);
int3 const_func __attribute__((overloadable)) rhadd(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) rhadd(uint3 x, uint3 y);
int4 const_func __attribute__((overloadable)) rhadd(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) rhadd(uint4 x, uint4 y);
int8 const_func __attribute__((overloadable)) rhadd(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) rhadd(uint8 x, uint8 y);
int16 const_func __attribute__((overloadable)) rhadd(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) rhadd(uint16 x, uint16 y);
long const_func __attribute__((overloadable)) rhadd(long x, long y);
ulong const_func __attribute__((overloadable)) rhadd(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) rhadd(long2 x, long2 y);
ulong2 const_func __attribute__((overloadable)) rhadd(ulong2 x, ulong2 y);
long3 const_func __attribute__((overloadable)) rhadd(long3 x, long3 y);
ulong3 const_func __attribute__((overloadable)) rhadd(ulong3 x, ulong3 y);
long4 const_func __attribute__((overloadable)) rhadd(long4 x, long4 y);
ulong4 const_func __attribute__((overloadable)) rhadd(ulong4 x, ulong4 y);
long8 const_func __attribute__((overloadable)) rhadd(long8 x, long8 y);
ulong8 const_func __attribute__((overloadable)) rhadd(ulong8 x, ulong8 y);
long16 const_func __attribute__((overloadable)) rhadd(long16 x, long16 y);
ulong16 const_func __attribute__((overloadable)) rhadd(ulong16 x, ulong16 y);

/**
 * Returns min(max(x, minval), maxval).
 * Results are undefined if minval > maxval.
 */
char const_func __attribute__((overloadable)) clamp(char x, char minval, char maxval);
uchar const_func __attribute__((overloadable)) clamp(uchar x, uchar minval, uchar maxval);
char2 const_func __attribute__((overloadable)) clamp(char2 x, char2 minval, char2 maxval);
uchar2 const_func __attribute__((overloadable)) clamp(uchar2 x, uchar2 minval, uchar2 maxval);
char3 const_func __attribute__((overloadable)) clamp(char3
```

```
x, char3 minval, char3 maxval);
uchar3 const_func __attribute__((overloadable)) clamp(uchar3
x, uchar3 minval, uchar3 maxval);
char4 const_func __attribute__((overloadable)) clamp(char4
x, char4 minval, char4 maxval);
uchar4 const_func __attribute__((overloadable)) clamp(uchar4
x, uchar4 minval, uchar4 maxval);
char8 const_func __attribute__((overloadable)) clamp(char8
x, char8 minval, char8 maxval);
uchar8 const_func __attribute__((overloadable)) clamp(uchar8
x, uchar8 minval, uchar8 maxval);
char16 const_func __attribute__((overloadable)) clamp(char16
x, char16 minval, char16 maxval);
uchar16 const_func __attribute__((overloadable)) clamp(uchar16
x, uchar16 minval, uchar16 maxval);
short const_func __attribute__((overloadable)) clamp(short
x, short minval, short maxval);
ushort const_func __attribute__((overloadable)) clamp(
ushort x, ushort minval, ushort maxval);
short2 const_func __attribute__((overloadable)) clamp(short2
x, short2 minval, short2 maxval);
ushort2 const_func __attribute__((overloadable)) clamp(ushort2
x, ushort2 minval, ushort2 maxval);
short3 const_func __attribute__((overloadable)) clamp(short3
x, short3 minval, short3 maxval);
ushort3 const_func __attribute__((overloadable)) clamp(ushort3
x, ushort3 minval, ushort3 maxval);
short4 const_func __attribute__((overloadable)) clamp(short4
x, short4 minval, short4 maxval);
ushort4 const_func __attribute__((overloadable)) clamp(ushort4
x, ushort4 minval, ushort4 maxval);
short8 const_func __attribute__((overloadable)) clamp(short8
x, short8 minval, short8 maxval);
ushort8 const_func __attribute__((overloadable)) clamp(ushort8
x, ushort8 minval, ushort8 maxval);
short16 const_func __attribute__((overloadable)) clamp(short16
x, short16 minval, short16 maxval);
ushort16 const_func __attribute__((overloadable)) clamp(ushort16
x, ushort16 minval, ushort16 maxval);
int const_func __attribute__((overloadable)) clamp(int x, int
minval, int maxval);
uint const_func __attribute__((overloadable)) clamp(
uint x, uint minval, uint maxval);
int2 const_func __attribute__((overloadable)) clamp(int2
x, int2 minval, int2 maxval);
uint2 const_func __attribute__((overloadable)) clamp(uint2
x, uint2 minval, uint2 maxval);
int3 const_func __attribute__((overloadable)) clamp(int3
x, int3 minval, int3 maxval);
uint3 const_func __attribute__((overloadable)) clamp(uint3
x, uint3 minval, uint3 maxval);
int4 const_func __attribute__((overloadable)) clamp(int4
x, int4 minval, int4 maxval);
uint4 const_func __attribute__((overloadable)) clamp(uint4
x, uint4 minval, uint4 maxval);
int8 const_func __attribute__((overloadable)) clamp(int8
x, int8 minval, int8 maxval);
uint8 const_func __attribute__((overloadable)) clamp(uint8
x, uint8 minval, uint8 maxval);
int16 const_func __attribute__((overloadable)) clamp(int16
x, int16 minval, int16 maxval);
uint16 const_func __attribute__((overloadable)) clamp(uint16
x, uint16 minval, uint16 maxval);
long const_func __attribute__((overloadable)) clamp(long
x, long minval, long maxval);
ulong const_func __attribute__((overloadable)) clamp(
ulong x, ulong minval, ulong maxval);
long2 const_func __attribute__((overloadable)) clamp(long2
x, long2 minval, long2 maxval);
ulong2 const_func __attribute__((overloadable)) clamp(ulong2
x, ulong2 minval, ulong2 maxval);
long3 const_func __attribute__((overloadable)) clamp(long3
x, long3 minval, long3 maxval);
ulong3 const_func __attribute__((overloadable)) clamp(ulong3
x, ulong3 minval, ulong3 maxval);
long4 const_func __attribute__((overloadable)) clamp(long4
x, long4 minval, long4 maxval);
ulong4 const_func __attribute__((overloadable)) clamp(ulong4
x, ulong4 minval, ulong4 maxval);
long8 const_func __attribute__((overloadable)) clamp(long8
x, long8 minval, long8 maxval);
ulong8 const_func __attribute__((overloadable)) clamp(ulong8
x, ulong8 minval, ulong8 maxval);
long16 const_func __attribute__((overloadable)) clamp(long16
x, long16 minval, long16 maxval);
ulong16 const_func __attribute__((overloadable)) clamp(ulong16
x, ulong16 minval, ulong16 maxval);
```

```
char const_func __attribute__((overloadable)) clamp(char x, char minval, char maxval);
uchar const_func __attribute__((overloadable)) clamp(uchar x, uchar minval, uchar maxval);
char2 const_func __attribute__((overloadable)) clamp(char2 x, char minval, char maxval);
uchar2 const_func __attribute__((overloadable)) clamp(uchar2 x, uchar minval, uchar maxval);
char3 const_func __attribute__((overloadable)) clamp(char3 x, char minval, char maxval);
uchar3 const_func __attribute__((overloadable)) clamp(uchar3 x, uchar minval, uchar maxval);
char4 const_func __attribute__((overloadable)) clamp(char4 x, char minval, char maxval);
uchar4 const_func __attribute__((overloadable)) clamp(uchar4 x, uchar minval, uchar maxval);
char8 const_func __attribute__((overloadable)) clamp(char8 x, char minval, char maxval);
uchar8 const_func __attribute__((overloadable)) clamp(uchar8 x, uchar minval, uchar maxval);
char16 const_func __attribute__((overloadable)) clamp(char16 x, char minval, char maxval);
uchar16 const_func __attribute__((overloadable)) clamp(uchar16 x, uchar minval, uchar maxval);
short const_func __attribute__((overloadable)) clamp(short x, short minval, short maxval);
ushort const_func __attribute__((overloadable)) clamp(ushort x, ushort minval, ushort maxval);
short2 const_func __attribute__((overloadable)) clamp(short2 x, short minval, short maxval);
ushort2 const_func __attribute__((overloadable)) clamp(ushort2 x, ushort minval, ushort maxval);
short3 const_func __attribute__((overloadable)) clamp(short3 x, short minval, short maxval);
ushort3 const_func __attribute__((overloadable)) clamp(ushort3 x, ushort minval, ushort maxval);
short4 const_func __attribute__((overloadable)) clamp(short4 x, short minval, short maxval);
ushort4 const_func __attribute__((overloadable)) clamp(ushort4 x, ushort minval, ushort maxval);
short8 const_func __attribute__((overloadable)) clamp(short8 x, short minval, short maxval);
ushort8 const_func __attribute__((overloadable)) clamp(ushort8 x, ushort minval, ushort maxval);
short16 const_func __attribute__((overloadable)) clamp(short16 x, short minval, short maxval);
ushort16 const_func __attribute__((overloadable)) clamp(ushort16 x, ushort minval, ushort maxval);
int const_func __attribute__((overloadable)) clamp(int x, int minval, int maxval);
uint const_func __attribute__((overloadable)) clamp(uint x, uint minval, uint maxval);
int2 const_func __attribute__((overloadable)) clamp(int2 x, int minval, int maxval);
uint2 const_func __attribute__((overloadable)) clamp(uint2 x, uint minval, uint maxval);
int3 const_func __attribute__((overloadable)) clamp(int3 x, int minval, int maxval);
uint3 const_func __attribute__((overloadable)) clamp(uint3 x, uint minval, uint maxval);
int4 const_func __attribute__((overloadable)) clamp(int4 x, int minval, int maxval);
uint4 const_func __attribute__((overloadable)) clamp(uint4 x, uint minval, uint maxval);
int8 const_func __attribute__((overloadable)) clamp(int8 x, int minval, int maxval);
uint8 const_func __attribute__((overloadable)) clamp(uint8 x, uint minval, uint maxval);
int16 const_func __attribute__((overloadable)) clamp(int16 x, int minval, int maxval);
uint16 const_func __attribute__((overloadable)) clamp(uint16 x, uint minval, uint maxval);
long const_func __attribute__((overloadable)) clamp(long x, long minval, long maxval);
ulong const_func __attribute__((overloadable)) clamp(ulong x, ulong minval, ulong maxval);
long2 const_func __attribute__((overloadable)) clamp(long2 x, long minval, long maxval);
ulong2 const_func __attribute__((overloadable)) clamp(ulong2 x, ulong minval, ulong maxval);
long3 const_func __attribute__((overloadable)) clamp(long3 x, long minval, long maxval);
ulong3 const_func __attribute__((overloadable)) clamp(ulong3 x, ulong minval, ulong maxval);
long4 const_func __attribute__((overloadable)) clamp(long4 x, long minval, long maxval);
ulong4 const_func __attribute__((overloadable)) clamp(ulong4
```

```
x, ulong minval, ulong maxval);
long8 const_func __attribute__((overloadable)) clamp(long8
x, long minval, long maxval);
ulong8 const_func __attribute__((overloadable)) clamp(ulong8
x, ulong minval, ulong maxval);
long16 const_func __attribute__((overloadable)) clamp(long16
x, long minval, long maxval);
ulong16 const_func __attribute__((overloadable)) clamp(ulong16
x, ulong minval, ulong maxval);

/**
 * Returns the number of leading 0-bits in x, starting
 * at the most significant bit position.
 */
char const_func __attribute__((overloadable)) clz(char x);
uchar const_func __attribute__((overloadable)) clz(
uchar x);
char2 const_func __attribute__((overloadable)) clz(char2
x);
uchar2 const_func __attribute__((overloadable)) clz(uchar2
x);
char3 const_func __attribute__((overloadable)) clz(char3
x);
uchar3 const_func __attribute__((overloadable)) clz(uchar3
x);
char4 const_func __attribute__((overloadable)) clz(char4
x);
uchar4 const_func __attribute__((overloadable)) clz(uchar4
x);
char8 const_func __attribute__((overloadable)) clz(char8
x);
uchar8 const_func __attribute__((overloadable)) clz(uchar8
x);
char16 const_func __attribute__((overloadable)) clz(char16
x);
uchar16 const_func __attribute__((overloadable)) clz(uchar16
x);
short const_func __attribute__((overloadable)) clz(short
x);
ushort const_func __attribute__((overloadable)) clz(
ushort x);
short2 const_func __attribute__((overloadable)) clz(short2
x);
ushort2 const_func __attribute__((overloadable)) clz(ushort2
x);
short3 const_func __attribute__((overloadable)) clz(short3
x);
ushort3 const_func __attribute__((overloadable)) clz(ushort3
x);
short4 const_func __attribute__((overloadable)) clz(short4
x);
ushort4 const_func __attribute__((overloadable)) clz(ushort4
x);
short8 const_func __attribute__((overloadable)) clz(short8
x);
ushort8 const_func __attribute__((overloadable)) clz(ushort8
x);
short16 const_func __attribute__((overloadable)) clz(short16
x);
ushort16 const_func __attribute__((overloadable)) clz(ushort16
x);
int const_func __attribute__((overloadable)) clz(int x);
uint const_func __attribute__((overloadable)) clz(
uint x);
int2 const_func __attribute__((overloadable)) clz(int2 x);
uint2 const_func __attribute__((overloadable)) clz(uint2
x);
int3 const_func __attribute__((overloadable)) clz(int3 x);
uint3 const_func __attribute__((overloadable)) clz(uint3
x);
int4 const_func __attribute__((overloadable)) clz(int4 x);
uint4 const_func __attribute__((overloadable)) clz(uint4
x);
int8 const_func __attribute__((overloadable)) clz(int8 x);
uint8 const_func __attribute__((overloadable)) clz(uint8
x);
int16 const_func __attribute__((overloadable)) clz(int16
x);
uint16 const_func __attribute__((overloadable)) clz(uint16
x);
long const_func __attribute__((overloadable)) clz(long x);
ulong const_func __attribute__((overloadable)) clz(
ulong x);
long2 const_func __attribute__((overloadable)) clz(long2
x);
ulong2 const_func __attribute__((overloadable)) clz(ulong2
x);
```

```
long3 const_func __attribute__((overloadable)) clz(long3 x);
ulong3 const_func __attribute__((overloadable)) clz(ulong3 x);
long4 const_func __attribute__((overloadable)) clz(long4 x);
ulong4 const_func __attribute__((overloadable)) clz(ulong4 x);
long8 const_func __attribute__((overloadable)) clz(long8 x);
ulong8 const_func __attribute__((overloadable)) clz(ulong8 x);
long16 const_func __attribute__((overloadable)) clz(long16 x);
ulong16 const_func __attribute__((overloadable)) clz(ulong16 x);

/**
 * Returns mul_hi(a, b) + c.
 */
char const_func __attribute__((overloadable)) mad_hi(char a, char b, char c);
uchar const_func __attribute__((overloadable)) mad_hi(uchar a, uchar b, uchar c);
char2 const_func __attribute__((overloadable)) mad_hi(char2 a, char2 b, char2 c);
uchar2 const_func __attribute__((overloadable)) mad_hi(uchar2 a, uchar2 b, uchar2 c);
char3 const_func __attribute__((overloadable)) mad_hi(char3 a, char3 b, char3 c);
uchar3 const_func __attribute__((overloadable)) mad_hi(uchar3 a, uchar3 b, uchar3 c);
char4 const_func __attribute__((overloadable)) mad_hi(char4 a, char4 b, char4 c);
uchar4 const_func __attribute__((overloadable)) mad_hi(uchar4 a, uchar4 b, uchar4 c);
char8 const_func __attribute__((overloadable)) mad_hi(char8 a, char8 b, char8 c);
uchar8 const_func __attribute__((overloadable)) mad_hi(uchar8 a, uchar8 b, uchar8 c);
char16 const_func __attribute__((overloadable)) mad_hi(char16 a, char16 b, char16 c);
uchar16 const_func __attribute__((overloadable)) mad_hi(uchar16 a, uchar16 b, uchar16 c);
short const_func __attribute__((overloadable)) mad_hi(short a, short b, short c);
ushort const_func __attribute__((overloadable)) mad_hi(ushort a, ushort b, ushort c);
short2 const_func __attribute__((overloadable)) mad_hi(short2 a, short2 b, short2 c);
ushort2 const_func __attribute__((overloadable)) mad_hi(ushort2 a, ushort2 b, ushort2 c);
short3 const_func __attribute__((overloadable)) mad_hi(short3 a, short3 b, short3 c);
ushort3 const_func __attribute__((overloadable)) mad_hi(ushort3 a, ushort3 b, ushort3 c);
short4 const_func __attribute__((overloadable)) mad_hi(short4 a, short4 b, short4 c);
ushort4 const_func __attribute__((overloadable)) mad_hi(ushort4 a, ushort4 b, ushort4 c);
short8 const_func __attribute__((overloadable)) mad_hi(short8 a, short8 b, short8 c);
ushort8 const_func __attribute__((overloadable)) mad_hi(ushort8 a, ushort8 b, ushort8 c);
short16 const_func __attribute__((overloadable)) mad_hi(short16 a, short16 b, short16 c);
ushort16 const_func __attribute__((overloadable)) mad_hi(ushort16 a, ushort16 b, ushort16 c);
int const_func __attribute__((overloadable)) mad_hi(int a, int b, int c);
uint const_func __attribute__((overloadable)) mad_hi(uint a, uint b, uint c);
int2 const_func __attribute__((overloadable)) mad_hi(int2 a, int2 b, int2 c);
uint2 const_func __attribute__((overloadable)) mad_hi(uint2 a, uint2 b, uint2 c);
int3 const_func __attribute__((overloadable)) mad_hi(int3 a, int3 b, int3 c);
uint3 const_func __attribute__((overloadable)) mad_hi(uint3 a, uint3 b, uint3 c);
int4 const_func __attribute__((overloadable)) mad_hi(int4 a, int4 b, int4 c);
uint4 const_func __attribute__((overloadable)) mad_hi(uint4 a, uint4 b, uint4 c);
int8 const_func __attribute__((overloadable)) mad_hi(int8 a, int8 b, int8 c);
uint8 const_func __attribute__((overloadable)) mad_hi(uint8
```

```
a, uint8 b, uint8 c);
int16 const_func __attribute__((overloadable)) mad_hi(int16
a, int16 b, int16 c);
uint16 const_func __attribute__((overloadable)) mad_hi(uint16
a, uint16 b, uint16 c);
long const_func __attribute__((overloadable)) mad_hi(long
a, long b, long c);
ulong const_func __attribute__((overloadable)) mad_hi(
ulong a, ulong b, ulong c);
long2 const_func __attribute__((overloadable)) mad_hi(long2
a, long2 b, long2 c);
ulong2 const_func __attribute__((overloadable)) mad_hi(ulong2
a, ulong2 b, ulong2 c);
long3 const_func __attribute__((overloadable)) mad_hi(long3
a, long3 b, long3 c);
ulong3 const_func __attribute__((overloadable)) mad_hi(ulong3
a, ulong3 b, ulong3 c);
long4 const_func __attribute__((overloadable)) mad_hi(long4
a, long4 b, long4 c);
ulong4 const_func __attribute__((overloadable)) mad_hi(ulong4
a, ulong4 b, ulong4 c);
long8 const_func __attribute__((overloadable)) mad_hi(long8
a, long8 b, long8 c);
ulong8 const_func __attribute__((overloadable)) mad_hi(ulong8
a, ulong8 b, ulong8 c);
long16 const_func __attribute__((overloadable)) mad_hi(long16
a, long16 b, long16 c);
ulong16 const_func __attribute__((overloadable)) mad_hi(ulong16
a, ulong16 b, ulong16 c);

/**
 * Returns a * b + c and saturates the result.
 */
char const_func __attribute__((overloadable)) mad_sat(char
a, char b, char c);
uchar const_func __attribute__((overloadable)) mad_sat(
uchar a, uchar b, uchar c);
char2 const_func __attribute__((overloadable)) mad_sat(char2
a, char2 b, char2 c);
uchar2 const_func __attribute__((overloadable)) mad_sat(uchar2
a, uchar2 b, uchar2 c);
char3 const_func __attribute__((overloadable)) mad_sat(char3
a, char3 b, char3 c);
uchar3 const_func __attribute__((overloadable)) mad_sat(uchar3
a, uchar3 b, uchar3 c);
char4 const_func __attribute__((overloadable)) mad_sat(char4
a, char4 b, char4 c);
uchar4 const_func __attribute__((overloadable)) mad_sat(uchar4
a, uchar4 b, uchar4 c);
char8 const_func __attribute__((overloadable)) mad_sat(char8
a, char8 b, char8 c);
uchar8 const_func __attribute__((overloadable)) mad_sat(uchar8
a, uchar8 b, uchar8 c);
char16 const_func __attribute__((overloadable)) mad_sat(char16
a, char16 b, char16 c);
uchar16 const_func __attribute__((overloadable)) mad_sat(uchar16
a, uchar16 b, uchar16 c);
short const_func __attribute__((overloadable)) mad_sat(short
a, short b, short c);
ushort const_func __attribute__((overloadable)) mad_sat(
ushort a, ushort b, ushort c);
short2 const_func __attribute__((overloadable)) mad_sat(short2
a, short2 b, short2 c);
ushort2 const_func __attribute__((overloadable)) mad_sat(ushort2
a, ushort2 b, ushort2 c);
short3 const_func __attribute__((overloadable)) mad_sat(short3
a, short3 b, short3 c);
ushort3 const_func __attribute__((overloadable)) mad_sat(ushort3
a, ushort3 b, ushort3 c);
short4 const_func __attribute__((overloadable)) mad_sat(short4
a, short4 b, short4 c);
ushort4 const_func __attribute__((overloadable)) mad_sat(ushort4
a, ushort4 b, ushort4 c);
short8 const_func __attribute__((overloadable)) mad_sat(short8
a, short8 b, short8 c);
ushort8 const_func __attribute__((overloadable)) mad_sat(ushort8
a, ushort8 b, ushort8 c);
short16 const_func __attribute__((overloadable)) mad_sat(short16
a, short16 b, short16 c);
ushort16 const_func __attribute__((overloadable)) mad_sat(ushort16
a, ushort16 b, ushort16 c);
int const_func __attribute__((overloadable)) mad_sat(int
a, int b, int c);
uint const_func __attribute__((overloadable)) mad_sat(
uint a, uint b, uint c);
int2 const_func __attribute__((overloadable)) mad_sat(int2
a, int2 b, int2 c);
```

```
uint2 const_func __attribute__((overloadable)) mad_sat(uint2 a, uint2 b, uint2 c);
int3 const_func __attribute__((overloadable)) mad_sat(int3 a, int3 b, int3 c);
uint3 const_func __attribute__((overloadable)) mad_sat(uint3 a, uint3 b, uint3 c);
int4 const_func __attribute__((overloadable)) mad_sat(int4 a, int4 b, int4 c);
uint4 const_func __attribute__((overloadable)) mad_sat(uint4 a, uint4 b, uint4 c);
int8 const_func __attribute__((overloadable)) mad_sat(int8 a, int8 b, int8 c);
uint8 const_func __attribute__((overloadable)) mad_sat(uint8 a, uint8 b, uint8 c);
int16 const_func __attribute__((overloadable)) mad_sat(int16 a, int16 b, int16 c);
uint16 const_func __attribute__((overloadable)) mad_sat(uint16 a, uint16 b, uint16 c);
long const_func __attribute__((overloadable)) mad_sat(long a, long b, long c);
ulong const_func __attribute__((overloadable)) mad_sat(ulong a, ulong b, ulong c);
long2 const_func __attribute__((overloadable)) mad_sat(long2 a, long2 b, long2 c);
ulong2 const_func __attribute__((overloadable)) mad_sat(ulong2 a, ulong2 b, ulong2 c);
long3 const_func __attribute__((overloadable)) mad_sat(long3 a, long3 b, long3 c);
ulong3 const_func __attribute__((overloadable)) mad_sat(ulong3 a, ulong3 b, ulong3 c);
long4 const_func __attribute__((overloadable)) mad_sat(long4 a, long4 b, long4 c);
ulong4 const_func __attribute__((overloadable)) mad_sat(ulong4 a, ulong4 b, ulong4 c);
long8 const_func __attribute__((overloadable)) mad_sat(long8 a, long8 b, long8 c);
ulong8 const_func __attribute__((overloadable)) mad_sat(ulong8 a, ulong8 b, ulong8 c);
long16 const_func __attribute__((overloadable)) mad_sat(long16 a, long16 b, long16 c);
ulong16 const_func __attribute__((overloadable)) mad_sat(ulong16 a, ulong16 b, ulong16 c);

/**
 * Returns y if x < y, otherwise it returns x.
 */
char const_func __attribute__((overloadable)) max(char x, char y);
uchar const_func __attribute__((overloadable)) max(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) max(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) max(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) max(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) max(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) max(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) max(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) max(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) max(uchar8 x, uchar8 y);
char16 const_func __attribute__((overloadable)) max(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) max(uchar16 x, uchar16 y);
short const_func __attribute__((overloadable)) max(short x, short y);
ushort const_func __attribute__((overloadable)) max(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) max(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) max(ushort2 x, ushort2 y);
short3 const_func __attribute__((overloadable)) max(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) max(ushort3 x, ushort3 y);
short4 const_func __attribute__((overloadable)) max(short4 x, short4 y);
ushort4 const_func __attribute__((overloadable)) max(ushort4 x, ushort4 y);
short8 const_func __attribute__((overloadable)) max(short8
```

```
x, short8 y);
ushort8 const_func __attribute__((overloadable)) max(ushort8
x, ushort8 y);
short16 const_func __attribute__((overloadable)) max(short16
x, short16 y);
ushort16 const_func __attribute__((overloadable)) max(ushort16
x, ushort16 y);
int const_func __attribute__((overloadable)) max(int x, int
y);
uint const_func __attribute__((overloadable)) max(
uint x, uint y);
int2 const_func __attribute__((overloadable)) max(int2 x, int2
y);
uint2 const_func __attribute__((overloadable)) max(uint2
x, uint2 y);
int3 const_func __attribute__((overloadable)) max(int3 x, int3
y);
uint3 const_func __attribute__((overloadable)) max(uint3
x, uint3 y);
int4 const_func __attribute__((overloadable)) max(int4 x, int4
y);
uint4 const_func __attribute__((overloadable)) max(uint4
x, uint4 y);
int8 const_func __attribute__((overloadable)) max(int8 x, int8
y);
uint8 const_func __attribute__((overloadable)) max(uint8
x, uint8 y);
int16 const_func __attribute__((overloadable)) max(int16
x, int16 y);
uint16 const_func __attribute__((overloadable)) max(uint16
x, uint16 y);
long const_func __attribute__((overloadable)) max(long x, long
y);
ulong const_func __attribute__((overloadable)) max(
ulong x, ulong y);
long2 const_func __attribute__((overloadable)) max(long2
x, long2 y);
ulong2 const_func __attribute__((overloadable)) max(ulong2
x, ulong2 y);
long3 const_func __attribute__((overloadable)) max(long3
x, long3 y);
ulong3 const_func __attribute__((overloadable)) max(ulong3
x, ulong3 y);
long4 const_func __attribute__((overloadable)) max(long4
x, long4 y);
ulong4 const_func __attribute__((overloadable)) max(ulong4
x, ulong4 y);
long8 const_func __attribute__((overloadable)) max(long8
x, long8 y);
ulong8 const_func __attribute__((overloadable)) max(ulong8
x, ulong8 y);
long16 const_func __attribute__((overloadable)) max(long16
x, long16 y);
ulong16 const_func __attribute__((overloadable)) max(ulong16
x, ulong16 y);
char const_func __attribute__((overloadable)) max(char x, char
y);
uchar const_func __attribute__((overloadable)) max(
uchar x, uchar y);
char2 const_func __attribute__((overloadable)) max(char2
x, char y);
uchar2 const_func __attribute__((overloadable)) max(uchar2
x, uchar y);
char3 const_func __attribute__((overloadable)) max(char3
x, char y);
uchar3 const_func __attribute__((overloadable)) max(uchar3
x, uchar y);
char4 const_func __attribute__((overloadable)) max(char4
x, char y);
uchar4 const_func __attribute__((overloadable)) max(uchar4
x, uchar y);
char8 const_func __attribute__((overloadable)) max(char8
x, char y);
uchar8 const_func __attribute__((overloadable)) max(uchar8
x, uchar y);
char16 const_func __attribute__((overloadable)) max(char16
x, char y);
uchar16 const_func __attribute__((overloadable)) max(uchar16
x, uchar y);
short const_func __attribute__((overloadable)) max(short
x, short y);
ushort const_func __attribute__((overloadable)) max(
ushort x, ushort y);
short2 const_func __attribute__((overloadable)) max(short2
x, short y);
ushort2 const_func __attribute__((overloadable)) max(ushort2
x, ushort y);
```

```
short3 const_func __attribute__((overloadable)) max(short3 x, short y);
ushort3 const_func __attribute__((overloadable)) max(ushort3 x, ushort y);
short4 const_func __attribute__((overloadable)) max(short4 x, short y);
ushort4 const_func __attribute__((overloadable)) max(ushort4 x, ushort y);
short8 const_func __attribute__((overloadable)) max(short8 x, short y);
ushort8 const_func __attribute__((overloadable)) max(ushort8 x, ushort y);
short16 const_func __attribute__((overloadable)) max(short16 x, short y);
ushort16 const_func __attribute__((overloadable)) max(ushort16 x, ushort y);
int const_func __attribute__((overloadable)) max(int x, int y);
uint const_func __attribute__((overloadable)) max(uint x, uint y);
int2 const_func __attribute__((overloadable)) max(int2 x, int y);
uint2 const_func __attribute__((overloadable)) max(uint2 x, uint y);
int3 const_func __attribute__((overloadable)) max(int3 x, int y);
uint3 const_func __attribute__((overloadable)) max(uint3 x, uint y);
int4 const_func __attribute__((overloadable)) max(int4 x, int y);
uint4 const_func __attribute__((overloadable)) max(uint4 x, uint y);
int8 const_func __attribute__((overloadable)) max(int8 x, int y);
uint8 const_func __attribute__((overloadable)) max(uint8 x, uint y);
int16 const_func __attribute__((overloadable)) max(int16 x, int y);
uint16 const_func __attribute__((overloadable)) max(uint16 x, uint y);
long const_func __attribute__((overloadable)) max(long x, long y);
ulong const_func __attribute__((overloadable)) max(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) max(long2 x, long y);
ulong2 const_func __attribute__((overloadable)) max(ulong2 x, ulong y);
long3 const_func __attribute__((overloadable)) max(long3 x, long y);
ulong3 const_func __attribute__((overloadable)) max(ulong3 x, ulong y);
long4 const_func __attribute__((overloadable)) max(long4 x, long y);
ulong4 const_func __attribute__((overloadable)) max(ulong4 x, ulong y);
long8 const_func __attribute__((overloadable)) max(long8 x, long y);
ulong8 const_func __attribute__((overloadable)) max(ulong8 x, ulong y);
long16 const_func __attribute__((overloadable)) max(long16 x, long y);
ulong16 const_func __attribute__((overloadable)) max(ulong16 x, ulong y);

/**
 * Returns y if y < x, otherwise it returns x.
 */
char const_func __attribute__((overloadable)) min(char x, char y);
uchar const_func __attribute__((overloadable)) min(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) min(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) min(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) min(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) min(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) min(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) min(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) min(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) min(uchar8
```

```
x, uchar8 y);
char16 const_func __attribute__((overloadable)) min(char16
x, char16 y);
uchar16 const_func __attribute__((overloadable)) min(uchar16
x, uchar16 y);
short const_func __attribute__((overloadable)) min(short
x, short y);
ushort const_func __attribute__((overloadable)) min(
ushort x, ushort y);
short2 const_func __attribute__((overloadable)) min(short2
x, short2 y);
ushort2 const_func __attribute__((overloadable)) min(ushort2
x, ushort2 y);
short3 const_func __attribute__((overloadable)) min(short3
x, short3 y);
ushort3 const_func __attribute__((overloadable)) min(ushort3
x, ushort3 y);
short4 const_func __attribute__((overloadable)) min(short4
x, short4 y);
ushort4 const_func __attribute__((overloadable)) min(ushort4
x, ushort4 y);
short8 const_func __attribute__((overloadable)) min(short8
x, short8 y);
ushort8 const_func __attribute__((overloadable)) min(ushort8
x, ushort8 y);
short16 const_func __attribute__((overloadable)) min(short16
x, short16 y);
ushort16 const_func __attribute__((overloadable)) min(ushort16
x, ushort16 y);
int const_func __attribute__((overloadable)) min(int x, int
y);
uint const_func __attribute__((overloadable)) min(
uint x, uint y);
int2 const_func __attribute__((overloadable)) min(int2
x, int2 y);
uint2 const_func __attribute__((overloadable)) min(uint2
x, uint2 y);
int3 const_func __attribute__((overloadable)) min(int3
x, int3 y);
uint3 const_func __attribute__((overloadable)) min(uint3
x, uint3 y);
int4 const_func __attribute__((overloadable)) min(int4
x, int4 y);
uint4 const_func __attribute__((overloadable)) min(uint4
x, uint4 y);
int8 const_func __attribute__((overloadable)) min(int8
x, int8 y);
uint8 const_func __attribute__((overloadable)) min(uint8
x, uint8 y);
int16 const_func __attribute__((overloadable)) min(int16
x, int16 y);
uint16 const_func __attribute__((overloadable)) min(uint16
x, uint16 y);
long const_func __attribute__((overloadable)) min(long
x, long y);
ulong const_func __attribute__((overloadable)) min(
ulong x, ulong y);
long2 const_func __attribute__((overloadable)) min(long2
x, long2 y);
ulong2 const_func __attribute__((overloadable)) min(ulong2
x, ulong2 y);
long3 const_func __attribute__((overloadable)) min(long3
x, long3 y);
ulong3 const_func __attribute__((overloadable)) min(ulong3
x, ulong3 y);
long4 const_func __attribute__((overloadable)) min(long4
x, long4 y);
ulong4 const_func __attribute__((overloadable)) min(ulong4
x, ulong4 y);
long8 const_func __attribute__((overloadable)) min(long8
x, long8 y);
ulong8 const_func __attribute__((overloadable)) min(ulong8
x, ulong8 y);
long16 const_func __attribute__((overloadable)) min(long16
x, long16 y);
ulong16 const_func __attribute__((overloadable)) min(ulong16
x, ulong16 y);
char const_func __attribute__((overloadable)) min(char
x, char y);
uchar const_func __attribute__((overloadable)) min(
uchar x, uchar y);
char2 const_func __attribute__((overloadable)) min(char2
x, char y);
uchar2 const_func __attribute__((overloadable)) min(uchar2
x, uchar y);
char3 const_func __attribute__((overloadable)) min(char3
x, char y);
```

```
uchar3 const_func __attribute__((overloadable)) min(uchar3 x, uchar y);
char4 const_func __attribute__((overloadable)) min(char4 x, char y);
uchar4 const_func __attribute__((overloadable)) min(uchar4 x, uchar y);
char8 const_func __attribute__((overloadable)) min(char8 x, char y);
uchar8 const_func __attribute__((overloadable)) min(uchar8 x, uchar y);
char16 const_func __attribute__((overloadable)) min(char16 x, char y);
uchar16 const_func __attribute__((overloadable)) min(uchar16 x, uchar y);
short const_func __attribute__((overloadable)) min(short x, short y);
ushort const_func __attribute__((overloadable)) min(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) min(short2 x, short y);
ushort2 const_func __attribute__((overloadable)) min(ushort2 x, ushort y);
short3 const_func __attribute__((overloadable)) min(short3 x, short y);
ushort3 const_func __attribute__((overloadable)) min(ushort3 x, ushort y);
short4 const_func __attribute__((overloadable)) min(short4 x, short y);
ushort4 const_func __attribute__((overloadable)) min(ushort4 x, ushort y);
short8 const_func __attribute__((overloadable)) min(short8 x, short y);
ushort8 const_func __attribute__((overloadable)) min(ushort8 x, ushort y);
short16 const_func __attribute__((overloadable)) min(short16 x, short y);
ushort16 const_func __attribute__((overloadable)) min(ushort16 x, ushort y);
int const_func __attribute__((overloadable)) min(int x, int y);
uint const_func __attribute__((overloadable)) min(uint x, uint y);
int2 const_func __attribute__((overloadable)) min(int2 x, int y);
uint2 const_func __attribute__((overloadable)) min(uint2 x, uint y);
int3 const_func __attribute__((overloadable)) min(int3 x, int y);
uint3 const_func __attribute__((overloadable)) min(uint3 x, uint y);
int4 const_func __attribute__((overloadable)) min(int4 x, int y);
uint4 const_func __attribute__((overloadable)) min(uint4 x, uint y);
int8 const_func __attribute__((overloadable)) min(int8 x, int y);
uint8 const_func __attribute__((overloadable)) min(uint8 x, uint y);
int16 const_func __attribute__((overloadable)) min(int16 x, int y);
uint16 const_func __attribute__((overloadable)) min(uint16 x, uint y);
long const_func __attribute__((overloadable)) min(long x, long y);
ulong const_func __attribute__((overloadable)) min(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) min(long2 x, long y);
ulong2 const_func __attribute__((overloadable)) min(ulong2 x, ulong y);
long3 const_func __attribute__((overloadable)) min(long3 x, long y);
ulong3 const_func __attribute__((overloadable)) min(ulong3 x, ulong y);
long4 const_func __attribute__((overloadable)) min(long4 x, long y);
ulong4 const_func __attribute__((overloadable)) min(ulong4 x, ulong y);
long8 const_func __attribute__((overloadable)) min(long8 x, long y);
ulong8 const_func __attribute__((overloadable)) min(ulong8 x, ulong y);
long16 const_func __attribute__((overloadable)) min(long16 x, long y);
ulong16 const_func __attribute__((overloadable)) min(ulong16 x, ulong y);
```

```
/**
 * Computes x * y and returns the high half of the
 * product of x and y.
 */
char const_func __attribute__((overloadable)) mul_hi(char x, char y);
uchar const_func __attribute__((overloadable)) mul_hi(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) mul_hi(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) mul_hi(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) mul_hi(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) mul_hi(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) mul_hi(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) mul_hi(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) mul_hi(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) mul_hi(uchar8 x, uchar8 y);
char16 const_func __attribute__((overloadable)) mul_hi(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) mul_hi(uchar16 x, uchar16 y);
short const_func __attribute__((overloadable)) mul_hi(short x, short y);
ushort const_func __attribute__((overloadable)) mul_hi(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) mul_hi(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) mul_hi(ushort2 x, ushort2 y);
short3 const_func __attribute__((overloadable)) mul_hi(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) mul_hi(ushort3 x, ushort3 y);
short4 const_func __attribute__((overloadable)) mul_hi(short4 x, short4 y);
ushort4 const_func __attribute__((overloadable)) mul_hi(ushort4 x, ushort4 y);
short8 const_func __attribute__((overloadable)) mul_hi(short8 x, short8 y);
ushort8 const_func __attribute__((overloadable)) mul_hi(ushort8 x, ushort8 y);
short16 const_func __attribute__((overloadable)) mul_hi(short16 x, short16 y);
ushort16 const_func __attribute__((overloadable)) mul_hi(ushort16 x, ushort16 y);
int const_func __attribute__((overloadable)) mul_hi(int x, int y);
uint const_func __attribute__((overloadable)) mul_hi(uint x, uint y);
int2 const_func __attribute__((overloadable)) mul_hi(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) mul_hi(uint2 x, uint2 y);
int3 const_func __attribute__((overloadable)) mul_hi(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) mul_hi(uint3 x, uint3 y);
int4 const_func __attribute__((overloadable)) mul_hi(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) mul_hi(uint4 x, uint4 y);
int8 const_func __attribute__((overloadable)) mul_hi(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) mul_hi(uint8 x, uint8 y);
int16 const_func __attribute__((overloadable)) mul_hi(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) mul_hi(uint16 x, uint16 y);
long const_func __attribute__((overloadable)) mul_hi(long x, long y);
ulong const_func __attribute__((overloadable)) mul_hi(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) mul_hi(long2 x, long2 y);
ulong2 const_func __attribute__((overloadable)) mul_hi(ulong2 x, ulong2 y);
long3 const_func __attribute__((overloadable)) mul_hi(long3 x, long3 y);
ulong3 const_func __attribute__((overloadable)) mul_hi(ulong3
```

```
x, ulong3 y);
long4 const_func __attribute__((overloadable)) mul_hi(long4
x, long4 y);
ulong4 const_func __attribute__((overloadable)) mul_hi(ulong4
x, ulong4 y);
long8 const_func __attribute__((overloadable)) mul_hi(long8
x, long8 y);
ulong8 const_func __attribute__((overloadable)) mul_hi(ulong8
x, ulong8 y);
long16 const_func __attribute__((overloadable)) mul_hi(long16
x, long16 y);
ulong16 const_func __attribute__((overloadable)) mul_hi(ulong16
x, ulong16 y);

/**
 * For each element in v, the bits are shifted left by
 * the number of bits given by the corresponding
 * element in i (subject to usual shift modulo rules
 * described in section 6.3). Bits shifted off the left
 * side of the element are shifted back in from the
 * right.
 */
char const_func __attribute__((overloadable)) rotate(char v, char
i);
uchar const_func __attribute__((overloadable)) rotate(
uchar v, uchar i);
char2 const_func __attribute__((overloadable)) rotate(char2 v, char2
i);
uchar2 const_func __attribute__((overloadable)) rotate(uchar2 v, uchar2
i);
char3 const_func __attribute__((overloadable)) rotate(char3 v, char3
i);
uchar3 const_func __attribute__((overloadable)) rotate(uchar3 v, uchar3
i);
char4 const_func __attribute__((overloadable)) rotate(char4 v, char4
i);
uchar4 const_func __attribute__((overloadable)) rotate(uchar4 v, uchar4
i);
char8 const_func __attribute__((overloadable)) rotate(char8 v, char8
i);
uchar8 const_func __attribute__((overloadable)) rotate(uchar8 v, uchar8
i);
char16 const_func __attribute__((overloadable)) rotate(char16 v, char16
i);
uchar16 const_func __attribute__((overloadable)) rotate(uchar16 v, uchar16
i);
short const_func __attribute__((overloadable)) rotate(short v, short
i);
ushort const_func __attribute__((overloadable)) rotate(
ushort v, ushort i);
short2 const_func __attribute__((overloadable)) rotate(short2 v, short2
i);
ushort2 const_func __attribute__((overloadable)) rotate(ushort2 v, ushort2
i);
short3 const_func __attribute__((overloadable)) rotate(short3 v, short3
i);
ushort3 const_func __attribute__((overloadable)) rotate(ushort3 v, ushort3
i);
short4 const_func __attribute__((overloadable)) rotate(short4 v, short4
i);
ushort4 const_func __attribute__((overloadable)) rotate(ushort4 v, ushort4
i);
short8 const_func __attribute__((overloadable)) rotate(short8 v, short8
i);
ushort8 const_func __attribute__((overloadable)) rotate(ushort8 v, ushort8
i);
short16 const_func __attribute__((overloadable)) rotate(short16 v, short16
i);
ushort16 const_func __attribute__((overloadable)) rotate(ushort16 v, ushort16
i);
int const_func __attribute__((overloadable)) rotate(int v, int
i);
uint const_func __attribute__((overloadable)) rotate(
uint v, uint i);
int2 const_func __attribute__((overloadable)) rotate(int2 v, int2
i);
uint2 const_func __attribute__((overloadable)) rotate(uint2 v, uint2
i);
int3 const_func __attribute__((overloadable)) rotate(int3 v, int3
i);
uint3 const_func __attribute__((overloadable)) rotate(uint3 v, uint3
i);
int4 const_func __attribute__((overloadable)) rotate(int4 v, int4
i);
uint4 const_func __attribute__((overloadable)) rotate(uint4 v, uint4
i);
int8 const_func __attribute__((overloadable)) rotate(int8 v, int8
```

```
i);
uint8 const_func __attribute__((overloadable)) rotate(uint8 v, uint8 i);
int16 const_func __attribute__((overloadable)) rotate(int16 v, int16 i);
uint16 const_func __attribute__((overloadable)) rotate(uint16 v, uint16 i);
long const_func __attribute__((overloadable)) rotate(long v, long i);
ulong const_func __attribute__((overloadable)) rotate(ulong v, ulong i);
long2 const_func __attribute__((overloadable)) rotate(long2 v, long2 i);
ulong2 const_func __attribute__((overloadable)) rotate(ulong2 v, ulong2 i);
long3 const_func __attribute__((overloadable)) rotate(long3 v, long3 i);
ulong3 const_func __attribute__((overloadable)) rotate(ulong3 v, ulong3 i);
long4 const_func __attribute__((overloadable)) rotate(long4 v, long4 i);
ulong4 const_func __attribute__((overloadable)) rotate(ulong4 v, ulong4 i);
long8 const_func __attribute__((overloadable)) rotate(long8 v, long8 i);
ulong8 const_func __attribute__((overloadable)) rotate(ulong8 v, ulong8 i);
long16 const_func __attribute__((overloadable)) rotate(long16 v, long16 i);
ulong16 const_func __attribute__((overloadable)) rotate(ulong16 v, ulong16 i);

/**
 * Returns x - y and saturates the result.
 */
char const_func __attribute__((overloadable)) sub_sat(char x, char y);
uchar const_func __attribute__((overloadable)) sub_sat(uchar x, uchar y);
char2 const_func __attribute__((overloadable)) sub_sat(char2 x, char2 y);
uchar2 const_func __attribute__((overloadable)) sub_sat(uchar2 x, uchar2 y);
char3 const_func __attribute__((overloadable)) sub_sat(char3 x, char3 y);
uchar3 const_func __attribute__((overloadable)) sub_sat(uchar3 x, uchar3 y);
char4 const_func __attribute__((overloadable)) sub_sat(char4 x, char4 y);
uchar4 const_func __attribute__((overloadable)) sub_sat(uchar4 x, uchar4 y);
char8 const_func __attribute__((overloadable)) sub_sat(char8 x, char8 y);
uchar8 const_func __attribute__((overloadable)) sub_sat(uchar8 x, uchar8 y);
char16 const_func __attribute__((overloadable)) sub_sat(char16 x, char16 y);
uchar16 const_func __attribute__((overloadable)) sub_sat(uchar16 x, uchar16 y);
short const_func __attribute__((overloadable)) sub_sat(short x, short y);
ushort const_func __attribute__((overloadable)) sub_sat(ushort x, ushort y);
short2 const_func __attribute__((overloadable)) sub_sat(short2 x, short2 y);
ushort2 const_func __attribute__((overloadable)) sub_sat(ushort2 x, ushort2 y);
short3 const_func __attribute__((overloadable)) sub_sat(short3 x, short3 y);
ushort3 const_func __attribute__((overloadable)) sub_sat(ushort3 x, ushort3 y);
short4 const_func __attribute__((overloadable)) sub_sat(short4 x, short4 y);
ushort4 const_func __attribute__((overloadable)) sub_sat(ushort4 x, ushort4 y);
short8 const_func __attribute__((overloadable)) sub_sat(short8 x, short8 y);
ushort8 const_func __attribute__((overloadable)) sub_sat(ushort8 x, ushort8 y);
short16 const_func __attribute__((overloadable)) sub_sat(short16 x, short16 y);
ushort16 const_func __attribute__((overloadable)) sub_sat(ushort16 x, ushort16 y);
int const_func __attribute__((overloadable)) sub_sat(int x, int y);
uint const_func __attribute__((overloadable)) sub_sat(uint x, uint y);
```

```
int2 const_func __attribute__((overloadable)) sub_sat(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) sub_sat(uint2 x, uint2 y);
int3 const_func __attribute__((overloadable)) sub_sat(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) sub_sat(uint3 x, uint3 y);
int4 const_func __attribute__((overloadable)) sub_sat(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) sub_sat(uint4 x, uint4 y);
int8 const_func __attribute__((overloadable)) sub_sat(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) sub_sat(uint8 x, uint8 y);
int16 const_func __attribute__((overloadable)) sub_sat(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) sub_sat(uint16 x, uint16 y);
long const_func __attribute__((overloadable)) sub_sat(long x, long y);
ulong const_func __attribute__((overloadable)) sub_sat(ulong x, ulong y);
long2 const_func __attribute__((overloadable)) sub_sat(long2 x, long2 y);
ulong2 const_func __attribute__((overloadable)) sub_sat(ulong2 x, ulong2 y);
long3 const_func __attribute__((overloadable)) sub_sat(long3 x, long3 y);
ulong3 const_func __attribute__((overloadable)) sub_sat(ulong3 x, ulong3 y);
long4 const_func __attribute__((overloadable)) sub_sat(long4 x, long4 y);
ulong4 const_func __attribute__((overloadable)) sub_sat(ulong4 x, ulong4 y);
long8 const_func __attribute__((overloadable)) sub_sat(long8 x, long8 y);
ulong8 const_func __attribute__((overloadable)) sub_sat(ulong8 x, ulong8 y);
long16 const_func __attribute__((overloadable)) sub_sat(long16 x, long16 y);
ulong16 const_func __attribute__((overloadable)) sub_sat(ulong16 x, ulong16 y);

/**
 * result[i] = ((short)hi[i] << 8) | lo[i]
 * result[i] = ((ushort)hi[i] << 8) | lo[i]
 */
short const_func __attribute__((overloadable)) upsample(char hi, uchar lo);
ushort const_func __attribute__((overloadable)) upsample (uchar hi, uchar lo);
short2 const_func __attribute__((overloadable)) upsample (char2 hi, uchar2 lo);
short3 const_func __attribute__((overloadable)) upsample (char3 hi, uchar3 lo);
short4 const_func __attribute__((overloadable)) upsample (char4 hi, uchar4 lo);
short8 const_func __attribute__((overloadable)) upsample (char8 hi, uchar8 lo);
short16 const_func __attribute__((overloadable)) upsample (char16 hi, uchar16 lo);
ushort2 const_func __attribute__((overloadable)) upsample (uchar2 hi, uchar2 lo);
ushort3 const_func __attribute__((overloadable)) upsample (uchar3 hi, uchar3 lo);
ushort4 const_func __attribute__((overloadable)) upsample (uchar4 hi, uchar4 lo);
ushort8 const_func __attribute__((overloadable)) upsample (uchar8 hi, uchar8 lo);
ushort16 const_func __attribute__((overloadable)) upsample (uchar16 hi, uchar16 lo);

/**
 * result[i] = ((int)hi[i] << 16) | lo[i]
 * result[i] = ((uint)hi[i] << 16) | lo[i]
 */
int const_func __attribute__((overloadable)) upsample(short hi, ushort lo);
uint const_func __attribute__((overloadable)) upsample (ushort hi, ushort lo);
int2 const_func __attribute__((overloadable)) upsample (short2 hi, ushort2 lo);
int3 const_func __attribute__((overloadable)) upsample (short3 hi, ushort3 lo);
int4 const_func __attribute__((overloadable)) upsample (short4 hi, ushort4
```

```
lo);
int8 const_func __attribute__((overloadable)) upsample (short8 hi, ushort8
lo);
int16 const_func __attribute__((overloadable)) upsample (short16 hi, ushort16
lo);
uint2 const_func __attribute__((overloadable)) upsample (ushort2 hi, ushort2
lo);
uint3 const_func __attribute__((overloadable)) upsample (ushort3 hi, ushort3
lo);
uint4 const_func __attribute__((overloadable)) upsample (ushort4 hi, ushort4
lo);
uint8 const_func __attribute__((overloadable)) upsample (ushort8 hi, ushort8
lo);
uint16 const_func __attribute__((overloadable)) upsample (ushort16 hi, ushort16
lo);
/**
 * result[i] = ((long)hi[i] << 32) | lo[i]
 * result[i] = ((ulong)hi[i] << 32) | lo[i]
 */
long const_func __attribute__((overloadable)) upsample(int hi,
uint lo);
ulong const_func __attribute__((overloadable)) upsample (
uint hi, uint lo);
long2 const_func __attribute__((overloadable)) upsample (int2 hi, uint2
lo);
long3 const_func __attribute__((overloadable)) upsample (int3 hi, uint3
lo);
long4 const_func __attribute__((overloadable)) upsample (int4 hi, uint4
lo);
long8 const_func __attribute__((overloadable)) upsample (int8 hi, uint8
lo);
long16 const_func __attribute__((overloadable)) upsample (int16 hi, uint16
lo);
ulong2 const_func __attribute__((overloadable)) upsample (uint2 hi, uint2
lo);
ulong3 const_func __attribute__((overloadable)) upsample (uint3 hi, uint3
lo);
ulong4 const_func __attribute__((overloadable)) upsample (uint4 hi, uint4
lo);
ulong8 const_func __attribute__((overloadable)) upsample (uint8 hi, uint8
lo);
ulong16 const_func __attribute__((overloadable)) upsample (uint16 hi, uint16
lo);

/*
 * popcount(x): returns the number of set bit in x
 */
char const_func __attribute__((overloadable)) popcount(char
x);
uchar const_func __attribute__((overloadable)) popcount(
uchar x);
char2 const_func __attribute__((overloadable)) popcount(char2
x);
uchar2 const_func __attribute__((overloadable)) popcount(uchar2
x);
char3 const_func __attribute__((overloadable)) popcount(char3
x);
uchar3 const_func __attribute__((overloadable)) popcount(uchar3
x);
char4 const_func __attribute__((overloadable)) popcount(char4
x);
uchar4 const_func __attribute__((overloadable)) popcount(uchar4
x);
char8 const_func __attribute__((overloadable)) popcount(char8
x);
uchar8 const_func __attribute__((overloadable)) popcount(uchar8
x);
char16 const_func __attribute__((overloadable)) popcount(char16
x);
uchar16 const_func __attribute__((overloadable)) popcount(uchar16
x);
short const_func __attribute__((overloadable)) popcount(short
x);
ushort const_func __attribute__((overloadable)) popcount(
ushort x);
short2 const_func __attribute__((overloadable)) popcount(short2
x);
ushort2 const_func __attribute__((overloadable)) popcount(ushort2
x);
short3 const_func __attribute__((overloadable)) popcount(short3
x);
ushort3 const_func __attribute__((overloadable)) popcount(ushort3
x);
short4 const_func __attribute__((overloadable)) popcount(short4
x);
ushort4 const_func __attribute__((overloadable)) popcount(ushort4
x);
```

```
short8 const_func __attribute__((overloadable)) popcount(short8 x);
ushort8 const_func __attribute__((overloadable)) popcount(ushort8 x);
short16 const_func __attribute__((overloadable)) popcount(short16 x);
ushort16 const_func __attribute__((overloadable)) popcount(ushort16 x);
int const_func __attribute__((overloadable)) popcount(int x);
uint const_func __attribute__((overloadable)) popcount(uint x);
int2 const_func __attribute__((overloadable)) popcount(int2 x);
uint2 const_func __attribute__((overloadable)) popcount(uint2 x);
int3 const_func __attribute__((overloadable)) popcount(int3 x);
uint3 const_func __attribute__((overloadable)) popcount(uint3 x);
int4 const_func __attribute__((overloadable)) popcount(int4 x);
uint4 const_func __attribute__((overloadable)) popcount(uint4 x);
int8 const_func __attribute__((overloadable)) popcount(int8 x);
uint8 const_func __attribute__((overloadable)) popcount(uint8 x);
int16 const_func __attribute__((overloadable)) popcount(int16 x);
uint16 const_func __attribute__((overloadable)) popcount(uint16 x);
long const_func __attribute__((overloadable)) popcount(long x);
ulong const_func __attribute__((overloadable)) popcount(ulong x);
long2 const_func __attribute__((overloadable)) popcount(long2 x);
ulong2 const_func __attribute__((overloadable)) popcount(ulong2 x);
long3 const_func __attribute__((overloadable)) popcount(long3 x);
ulong3 const_func __attribute__((overloadable)) popcount(ulong3 x);
long4 const_func __attribute__((overloadable)) popcount(long4 x);
ulong4 const_func __attribute__((overloadable)) popcount(ulong4 x);
long8 const_func __attribute__((overloadable)) popcount(long8 x);
ulong8 const_func __attribute__((overloadable)) popcount(ulong8 x);
long16 const_func __attribute__((overloadable)) popcount(long16 x);
ulong16 const_func __attribute__((overloadable)) popcount(ulong16 x);

/**
 * Multiply two 24-bit integer values x and y and add
 * the 32-bit integer result to the 32-bit integer z.
 * Refer to definition of mul24 to see how the 24-bit
 * integer multiplication is performed.
 */
int const_func __attribute__((overloadable)) mad24(int x, int y, int z);
uint const_func __attribute__((overloadable)) mad24(uint x, uint y, uint z);
int2 const_func __attribute__((overloadable)) mad24(int2 x, int2 y, int2 z);
uint2 const_func __attribute__((overloadable)) mad24(uint2 x, uint2 y, uint2 z);
int3 const_func __attribute__((overloadable)) mad24(int3 x, int3 y, int3 z);
uint3 const_func __attribute__((overloadable)) mad24(uint3 x, uint3 y, uint3 z);
int4 const_func __attribute__((overloadable)) mad24(int4 x, int4 y, int4 z);
uint4 const_func __attribute__((overloadable)) mad24(uint4 x, uint4 y, uint4 z);
int8 const_func __attribute__((overloadable)) mad24(int8 x, int8 y, int8 z);
uint8 const_func __attribute__((overloadable)) mad24(uint8 x, uint8 y, uint8 z);
int16 const_func __attribute__((overloadable)) mad24(int16 x, int16 y, int16 z);
uint16 const_func __attribute__((overloadable)) mad24(uint16 x, uint16 y, uint16 z);
```

```
/**
 * Multiply two 24-bit integer values x and y. x and y
 * are 32-bit integers but only the low 24-bits are used
 * to perform the multiplication. mul24 should only
 * be used when values in x and y are in the range [-
 * 2^23, 2^23-1] if x and y are signed integers and in the
 * range [0, 2^24-1] if x and y are unsigned integers. If
 * x and y are not in this range, the multiplication
 * result is implementation-defined.
 */
int const_func __attribute__((overloadable)) mul24(int x, int y);
uint const_func __attribute__((overloadable)) mul24(uint x, uint y);
int2 const_func __attribute__((overloadable)) mul24(int2 x, int2 y);
uint2 const_func __attribute__((overloadable)) mul24(uint2 x, uint2 y);
int3 const_func __attribute__((overloadable)) mul24(int3 x, int3 y);
uint3 const_func __attribute__((overloadable)) mul24(uint3 x, uint3 y);
int4 const_func __attribute__((overloadable)) mul24(int4 x, int4 y);
uint4 const_func __attribute__((overloadable)) mul24(uint4 x, uint4 y);
int8 const_func __attribute__((overloadable)) mul24(int8 x, int8 y);
uint8 const_func __attribute__((overloadable)) mul24(uint8 x, uint8 y);
int16 const_func __attribute__((overloadable)) mul24(int16 x, int16 y);
uint16 const_func __attribute__((overloadable)) mul24(uint16 x, uint16 y);

// Common functions:

/**
 * Returns fmin(fmax(x, minval), maxval).
 * Results are undefined if minval > maxval.
 */
float const_func __attribute__((overloadable)) clamp(float x, float minval, float maxval);
float2 const_func __attribute__((overloadable)) clamp(float2 x, float2 minval, float2 maxval);
float3 const_func __attribute__((overloadable)) clamp(float3 x, float3 minval, float3 maxval);
float4 const_func __attribute__((overloadable)) clamp(float4 x, float4 minval, float4 maxval);
float8 const_func __attribute__((overloadable)) clamp(float8 x, float8 minval, float8 maxval);
float16 const_func __attribute__((overloadable)) clamp(float16 x, float16 minval, float16 maxval);
float2 const_func __attribute__((overloadable)) clamp(float2 x, float minval, float maxval);
float3 const_func __attribute__((overloadable)) clamp(float3 x, float minval, float maxval);
float4 const_func __attribute__((overloadable)) clamp(float4 x, float minval, float maxval);
float8 const_func __attribute__((overloadable)) clamp(float8 x, float minval, float maxval);
float16 const_func __attribute__((overloadable)) clamp(float16 x, float minval, float maxval);
double const_func __attribute__((overloadable)) clamp(double x, double minval, double maxval);
double2 const_func __attribute__((overloadable)) clamp(double2 x, double2 minval, double2 maxval);
double3 const_func __attribute__((overloadable)) clamp(double3 x, double3 minval, double3 maxval);
double4 const_func __attribute__((overloadable)) clamp(double4 x, double4 minval, double4 maxval);
double8 const_func __attribute__((overloadable)) clamp(double8 x, double8 minval, double8 maxval);
double16 const_func __attribute__((overloadable)) clamp(double16 x, double16 minval, double16 maxval);
double2 const_func __attribute__((overloadable)) clamp(double2 x, double minval, double maxval);
double3 const_func __attribute__((overloadable)) clamp(double3 x, double minval, double maxval);
double4 const_func __attribute__((overloadable)) clamp(double4 x, double minval, double maxval);
double8 const_func __attribute__((overloadable)) clamp(double8 x, double minval, double maxval);
double16 const_func __attribute__((overloadable)) clamp(double16 x, double minval, double maxval);
//half const_func __attribute__((overloadable)) clamp(half x, half minval, half maxval);
```

```
//half2 const_func __attribute__((overloadable)) clamp(half2 x, half2 minval, half2 maxval);
//half3 const_func __attribute__((overloadable)) clamp(half3 x, half3 minval, half3 maxval);
//half4 const_func __attribute__((overloadable)) clamp(half4 x, half4 minval, half4 maxval);
//half8 const_func __attribute__((overloadable)) clamp(half8 x, half8 minval, half8 maxval);
//half16 const_func __attribute__((overloadable)) clamp(half16 x, half16 minval, half16 maxval);
//half2 const_func __attribute__((overloadable)) clamp(half2 x, half minval, half maxval);
//half3 const_func __attribute__((overloadable)) clamp(half3 x, half minval, half maxval);
//half4 const_func __attribute__((overloadable)) clamp(half4 x, half minval, half maxval);
//half8 const_func __attribute__((overloadable)) clamp(half8 x, half minval, half maxval);
//half16 const_func __attribute__((overloadable)) clamp(half16 x, half minval, half maxval);

/**
 * Converts radians to degrees, i.e. (180 / PI) *
 * radians.
 */
float const_func __attribute__((overloadable)) degrees(float radians);
float2 const_func __attribute__((overloadable)) degrees(float2 radians);
float3 const_func __attribute__((overloadable)) degrees(float3 radians);
float4 const_func __attribute__((overloadable)) degrees(float4 radians);
float8 const_func __attribute__((overloadable)) degrees(float8 radians);
float16 const_func __attribute__((overloadable)) degrees(float16 radians);
double const_func __attribute__((overloadable)) degrees(double radians);
double2 const_func __attribute__((overloadable)) degrees(double2 radians);
double3 const_func __attribute__((overloadable)) degrees(double3 radians);
double4 const_func __attribute__((overloadable)) degrees(double4 radians);
double8 const_func __attribute__((overloadable)) degrees(double8 radians);
double16 const_func __attribute__((overloadable)) degrees(double16 radians);
//half const_func __attribute__((overloadable)) degrees(half radians);
//half2 const_func __attribute__((overloadable)) degrees(half2 radians);
//half3 const_func __attribute__((overloadable)) degrees(half3 radians);
//half4 const_func __attribute__((overloadable)) degrees(half4 radians);
//half8 const_func __attribute__((overloadable)) degrees(half8 radians);
//half16 const_func __attribute__((overloadable)) degrees(half16 radians);

/**
 * Returns y if x < y, otherwise it returns x. If x and y
 * are infinite or NaN, the return values are undefined.
 */
float const_func __attribute__((overloadable)) max(float x, float y);
float2 const_func __attribute__((overloadable)) max(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) max(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) max(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) max(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) max(float16 x, float16 y);
float2 const_func __attribute__((overloadable)) max(float2 x, float y);
float3 const_func __attribute__((overloadable)) max(float3 x, float y);
float4 const_func __attribute__((overloadable)) max(float4 x, float y);
float8 const_func __attribute__((overloadable)) max(float8 x, float y);
float16 const_func __attribute__((overloadable)) max(float16 x, float y);
double const_func __attribute__((overloadable)) max(double x, double y);
double2 const_func __attribute__((overloadable)) max(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) max(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) max(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) max(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) max(double16 x, double16 y);
double2 const_func __attribute__((overloadable)) max(double2 x, double y);
double3 const_func __attribute__((overloadable)) max(double3 x, double y);
double4 const_func __attribute__((overloadable)) max(double4 x, double y);
double8 const_func __attribute__((overloadable)) max(double8 x, double y);
double16 const_func __attribute__((overloadable)) max(double16 x, double y);
//half const_func __attribute__((overloadable)) max(half x, half y);
//half2 const_func __attribute__((overloadable)) max(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) max(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) max(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) max(half8 x, half8 y);
```

```
//half16 const_func __attribute__((overloadable)) max(half16 x, half16 y);
//half2 const_func __attribute__((overloadable)) max(half2 x, half y);
//half3 const_func __attribute__((overloadable)) max(half3 x, half y);
//half4 const_func __attribute__((overloadable)) max(half4 x, half y);
//half8 const_func __attribute__((overloadable)) max(half8 x, half y);
//half16 const_func __attribute__((overloadable)) max(half16 x, half y);

/**
 * Returns y if y < x, otherwise it returns x. If x and y
 * are infinite or NaN, the return values are undefined.
 */
float const_func __attribute__((overloadable)) min(float x, float y);
float2 const_func __attribute__((overloadable)) min(float2 x, float2 y);
float3 const_func __attribute__((overloadable)) min(float3 x, float3 y);
float4 const_func __attribute__((overloadable)) min(float4 x, float4 y);
float8 const_func __attribute__((overloadable)) min(float8 x, float8 y);
float16 const_func __attribute__((overloadable)) min(float16 x, float16 y);
float2 const_func __attribute__((overloadable)) min(float2 x, float y);
float3 const_func __attribute__((overloadable)) min(float3 x, float y);
float4 const_func __attribute__((overloadable)) min(float4 x, float y);
float8 const_func __attribute__((overloadable)) min(float8 x, float y);
float16 const_func __attribute__((overloadable)) min(float16 x, float y);
double const_func __attribute__((overloadable)) min(double x, double y);
double2 const_func __attribute__((overloadable)) min(double2 x, double2 y);
double3 const_func __attribute__((overloadable)) min(double3 x, double3 y);
double4 const_func __attribute__((overloadable)) min(double4 x, double4 y);
double8 const_func __attribute__((overloadable)) min(double8 x, double8 y);
double16 const_func __attribute__((overloadable)) min(double16 x, double16 y);
double2 const_func __attribute__((overloadable)) min(double2 x, double y);
double3 const_func __attribute__((overloadable)) min(double3 x, double y);
double4 const_func __attribute__((overloadable)) min(double4 x, double y);
double8 const_func __attribute__((overloadable)) min(double8 x, double y);
double16 const_func __attribute__((overloadable)) min(double16 x, double y);
//half const_func __attribute__((overloadable)) min(half x, half y);
//half2 const_func __attribute__((overloadable)) min(half2 x, half2 y);
//half3 const_func __attribute__((overloadable)) min(half3 x, half3 y);
//half4 const_func __attribute__((overloadable)) min(half4 x, half4 y);
//half8 const_func __attribute__((overloadable)) min(half8 x, half8 y);
//half16 const_func __attribute__((overloadable)) min(half16 x, half16 y);
//half2 const_func __attribute__((overloadable)) min(half2 x, half y);
//half3 const_func __attribute__((overloadable)) min(half3 x, half y);
//half4 const_func __attribute__((overloadable)) min(half4 x, half y);
//half8 const_func __attribute__((overloadable)) min(half8 x, half y);
//half16 const_func __attribute__((overloadable)) min(half16 x, half y);

/**
 * Returns the linear blend of x & y implemented as:
 * x + (y - x) * a
 * a must be a value in the range 0.0 ... 1.0. If a is not
 * in the range 0.0 ... 1.0, the return values are
 * undefined.
 */
float const_func __attribute__((overloadable)) mix(float x, float y, float a);
float2 const_func __attribute__((overloadable)) mix(float2 x, float2 y, float2 a);
float3 const_func __attribute__((overloadable)) mix(float3 x, float3 y, float3 a);
float4 const_func __attribute__((overloadable)) mix(float4 x, float4 y, float4 a);
float8 const_func __attribute__((overloadable)) mix(float8 x, float8 y, float8 a);
float16 const_func __attribute__((overloadable)) mix(float16 x, float16 y, float16 a);
float2 const_func __attribute__((overloadable)) mix(float2
```

```
x, float2 y, float a);
float3 const_func __attribute__((overloadable)) mix(float3
x, float3 y, float a);
float4 const_func __attribute__((overloadable)) mix(float4
x, float4 y, float a);
float8 const_func __attribute__((overloadable)) mix(float8
x, float8 y, float a);
float16 const_func __attribute__((overloadable)) mix(float16
x, float16 y, float a);
double const_func __attribute__((overloadable)) mix(double
x, double y, double a);
double2 const_func __attribute__((overloadable)) mix(double2
x, double2 y, double2 a);
double3 const_func __attribute__((overloadable)) mix(double3
x, double3 y, double3 a);
double4 const_func __attribute__((overloadable)) mix(double4
x, double4 y, double4 a);
double8 const_func __attribute__((overloadable)) mix(double8
x, double8 y, double8 a);
double16 const_func __attribute__((overloadable)) mix(double16
x, double16 y, double16 a);
double2 const_func __attribute__((overloadable)) mix(double2
x, double2 y, double a);
double3 const_func __attribute__((overloadable)) mix(double3
x, double3 y, double a);
double4 const_func __attribute__((overloadable)) mix(double4
x, double4 y, double a);
double8 const_func __attribute__((overloadable)) mix(double8
x, double8 y, double a);
double16 const_func __attribute__((overloadable)) mix(double16
x, double16 y, double a);
//half const_func __attribute__((overloadable)) mix(half x, half y, half a);
//half2 const_func __attribute__((overloadable)) mix(half2 x, half2 y, half2 a);
//half3 const_func __attribute__((overloadable)) mix(half3 x, half3 y, half3 a);
//half4 const_func __attribute__((overloadable)) mix(half4 x, half4 y, half4 a);
//half8 const_func __attribute__((overloadable)) mix(half8 x, half8 y, half8 a);
//half16 const_func __attribute__((overloadable)) mix(half16 x, half16 y, half16 a);
//half2 const_func __attribute__((overloadable)) mix(half2 x, half2 y, half a);
//half3 const_func __attribute__((overloadable)) mix(half3 x, half3 y, half a);
//half4 const_func __attribute__((overloadable)) mix(half4 x, half4 y, half a);
//half8 const_func __attribute__((overloadable)) mix(half8 x, half8 y, half a);
//half16 const_func __attribute__((overloadable)) mix(half16 x, half16 y, half a);

/**
 * Converts degrees to radians, i.e. (PI / 180) *
 * degrees.
 */
float const_func __attribute__((overloadable)) radians(float degrees);
float2 const_func __attribute__((overloadable)) radians(float2 degrees);
float3 const_func __attribute__((overloadable)) radians(float3 degrees);
float4 const_func __attribute__((overloadable)) radians(float4 degrees);
float8 const_func __attribute__((overloadable)) radians(float8 degrees);
float16 const_func __attribute__((overloadable)) radians(float16 degrees);
double const_func __attribute__((overloadable)) radians(double degrees);
double2 const_func __attribute__((overloadable)) radians(double2 degrees);
double3 const_func __attribute__((overloadable)) radians(double3 degrees);
double4 const_func __attribute__((overloadable)) radians(double4 degrees);
double8 const_func __attribute__((overloadable)) radians(double8 degrees);
double16 const_func __attribute__((overloadable)) radians(double16 degrees);
//half const_func __attribute__((overloadable)) radians(half degrees);
//half2 const_func __attribute__((overloadable)) radians(half2 degrees);
//half3 const_func __attribute__((overloadable)) radians(half3 degrees);
//half4 const_func __attribute__((overloadable)) radians(half4 degrees);
//half8 const_func __attribute__((overloadable)) radians(half8 degrees);
//half16 const_func __attribute__((overloadable)) radians(half16 degrees);

/**
 * Returns 0.0 if x < edge, otherwise it returns 1.0.
 */
float const_func __attribute__((overloadable)) step(float edge, float
x);
float2 const_func __attribute__((overloadable)) step(float2 edge, float2
x);
float3 const_func __attribute__((overloadable)) step(float3 edge, float3
x);
float4 const_func __attribute__((overloadable)) step(float4 edge, float4
x);
float8 const_func __attribute__((overloadable)) step(float8 edge, float8
x);
float16 const_func __attribute__((overloadable)) step(float16 edge, float16
x);
float2 const_func __attribute__((overloadable)) step(float edge, float2
x);
float3 const_func __attribute__((overloadable)) step(float edge, float3
x);
float4 const_func __attribute__((overloadable)) step(float edge, float4
x);
```

```
float8 const_func __attribute__((overloadable)) step(float edge, float8 x);
float16 const_func __attribute__((overloadable)) step(float edge, float16 x);
double const_func __attribute__((overloadable)) step(double edge, double x);
double2 const_func __attribute__((overloadable)) step(double2 edge, double2 x);
double3 const_func __attribute__((overloadable)) step(double3 edge, double3 x);
double4 const_func __attribute__((overloadable)) step(double4 edge, double4 x);
double8 const_func __attribute__((overloadable)) step(double8 edge, double8 x);
double16 const_func __attribute__((overloadable)) step(double16 edge, double16 x);
double const_func __attribute__((overloadable)) step(double edge, double x);
double2 const_func __attribute__((overloadable)) step(double edge, double2 x);
double3 const_func __attribute__((overloadable)) step(double edge, double3 x);
double4 const_func __attribute__((overloadable)) step(double edge, double4 x);
double8 const_func __attribute__((overloadable)) step(double edge, double8 x);
double16 const_func __attribute__((overloadable)) step(double edge, double16 x);
//half const_func __attribute__((overloadable)) step(half edge, half x);
//half2 const_func __attribute__((overloadable)) step(half2 edge, half2 x);
//half3 const_func __attribute__((overloadable)) step(half3 edge, half3 x);
//half4 const_func __attribute__((overloadable)) step(half4 edge, half4 x);
//half8 const_func __attribute__((overloadable)) step(half8 edge, half8 x);
//half16 const_func __attribute__((overloadable)) step(half16 edge, half16 x);
//half const_func __attribute__((overloadable)) step(half edge, half x);
//half2 const_func __attribute__((overloadable)) step(half edge, half2 x);
//half3 const_func __attribute__((overloadable)) step(half edge, half3 x);
//half4 const_func __attribute__((overloadable)) step(half edge, half4 x);
//half8 const_func __attribute__((overloadable)) step(half edge, half8 x);
//half16 const_func __attribute__((overloadable)) step(half edge, half16 x);

/**
 * Returns 0.0 if x <= edge0 and 1.0 if x >= edge1 and
 * performs smooth Hermite interpolation between 0
 * and 1when edge0 < x < edge1. This is useful in
 * cases where you would want a threshold function
 * with a smooth transition.
 * This is equivalent to:
 * gentype t;
 * t = clamp ((x - edge0) / (edge1 - edge0), 0, 1);
 * return t * t * (3 - 2 * t);
 * Results are undefined if edge0 >= edge1 or if x,
 * edge0 or edge1 is a NaN.
 */
float const_func __attribute__((overloadable)) smoothstep(float edge0, float edge1, float x);
float2 const_func __attribute__((overloadable)) smoothstep(float2 edge0, float2 edge1, float2 x);
float3 const_func __attribute__((overloadable)) smoothstep(float3 edge0, float3 edge1, float3 x);
float4 const_func __attribute__((overloadable)) smoothstep(float4 edge0, float4 edge1, float4 x);
float8 const_func __attribute__((overloadable)) smoothstep(float8 edge0, float8 edge1, float8 x);
float16 const_func __attribute__((overloadable)) smoothstep(float16 edge0, float16 edge1, float16 x);
float2 const_func __attribute__((overloadable)) smoothstep(float edge0, float edge1, float2 x);
float3 const_func __attribute__((overloadable)) smoothstep(float edge0, float edge1, float3 x);
float4 const_func __attribute__((overloadable)) smoothstep(float edge0, float edge1, float4 x);
float8 const_func __attribute__((overloadable)) smoothstep(float edge0, float edge1, float8 x);
float16 const_func __attribute__((overloadable)) smoothstep(float edge0, float edge1, float16 x);
double const_func __attribute__((overloadable)) smoothstep(double edge0, double edge1, double x);
double2 const_func __attribute__((overloadable)) smoothstep(double edge0, double edge1, double2 x);
double3 const_func __attribute__((overloadable)) smoothstep(double edge0, double edge1, double3 x);
double4 const_func __attribute__((overloadable)) smoothstep(double edge0, double edge1, double4 x);
double8 const_func __attribute__((overloadable)) smoothstep(double edge0, double edge1, double8 x);
double16 const_func __attribute__((overloadable)) smoothstep(double edge0, double
```

```
edge1, double16 x);
double const_func __attribute__((overloadable)) smoothstep(double edge0, double edge1, double x);
double2 const_func __attribute__((overloadable)) smoothstep(double2 edge0, double2 edge1, double2 x);
double3 const_func __attribute__((overloadable)) smoothstep(double3 edge0, double3 edge1, double3 x);
double4 const_func __attribute__((overloadable)) smoothstep(double4 edge0, double4 edge1, double4 x);
double8 const_func __attribute__((overloadable)) smoothstep(double8 edge0, double8 edge1, double8 x);
double16 const_func __attribute__((overloadable)) smoothstep(double16 edge0, double16 edge1, double16 x);
//half const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half x);
//half2 const_func __attribute__((overloadable)) smoothstep(half2 edge0, half2 edge1, half2 x);
//half3 const_func __attribute__((overloadable)) smoothstep(half3 edge0, half3 edge1, half3 x);
//half4 const_func __attribute__((overloadable)) smoothstep(half4 edge0, half4 edge1, half4 x);
//half8 const_func __attribute__((overloadable)) smoothstep(half8 edge0, half8 edge1, half8 x);
//half16 const_func __attribute__((overloadable)) smoothstep(half16 edge0, half16 edge1, half16 x);
//half const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half x);
//half2 const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half2 x);
//half3 const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half3 x);
//half4 const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half4 x);
//half8 const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half8 x);
//half16 const_func __attribute__((overloadable)) smoothstep(half edge0, half edge1, half16 x);

/**
 * Returns 1.0 if x > 0, -0.0 if x = -0.0, +0.0 if x =
 * +0.0, or -1.0 if x < 0. Returns 0.0 if x is a NaN.
 */
float const_func __attribute__((overloadable)) sign(float x);
float2 const_func __attribute__((overloadable)) sign(float2 x);
float3 const_func __attribute__((overloadable)) sign(float3 x);
float4 const_func __attribute__((overloadable)) sign(float4 x);
float8 const_func __attribute__((overloadable)) sign(float8 x);
float16 const_func __attribute__((overloadable)) sign(float16 x);
double const_func __attribute__((overloadable)) sign(double x);
double2 const_func __attribute__((overloadable)) sign(double2 x);
double3 const_func __attribute__((overloadable)) sign(double3 x);
double4 const_func __attribute__((overloadable)) sign(double4 x);
double8 const_func __attribute__((overloadable)) sign(double8 x);
double16 const_func __attribute__((overloadable)) sign(double16 x);
//half const_func __attribute__((overloadable)) sign(half x);
//half2 const_func __attribute__((overloadable)) sign(half2 x);
//half3 const_func __attribute__((overloadable)) sign(half3 x);
//half4 const_func __attribute__((overloadable)) sign(half4 x);
//half8 const_func __attribute__((overloadable)) sign(half8 x);
//half16 const_func __attribute__((overloadable)) sign(half16 x);

// Geometric functions:

/**
 * Returns the cross product of p0.xyz and p1.xyz. The
 * w component of float4 result returned will be 0.0.
 */
float4 const_func __attribute__((overloadable)) cross(float4 p0, float4 p1);
float3 const_func __attribute__((overloadable)) cross (float3 p0, float3 p1);
double4 const_func __attribute__((overloadable)) cross(double4 p0, double4 p1);
double3 const_func __attribute__((overloadable)) cross(double3 p0, double3 p1);
//half4 const_func __attribute__((overloadable)) cross(half4 p0, half4 p1);
//half3 const_func __attribute__((overloadable)) cross(half3 p0, half3 p1);

/**
 * Compute dot product.
 */
float const_func __attribute__((overloadable)) dot(float p0, float p1);
float const_func __attribute__((overloadable)) dot(float2 p0, float2 p1);
float const_func __attribute__((overloadable)) dot(float3 p0, float3 p1);
```

```
float const_func __attribute__((overloadable)) dot(float4 p0, float4 p1);
double const_func __attribute__((overloadable)) dot(double p0, double p1);
double const_func __attribute__((overloadable)) dot(double2 p0, double2 p1);
double const_func __attribute__((overloadable)) dot(double3 p0, double3 p1);
double const_func __attribute__((overloadable)) dot(double4 p0, double4 p1);
//half const_func __attribute__((overloadable)) dot(half p0, half p1);
//half const_func __attribute__((overloadable)) dot(half2 p0, half2 p1);
//half const_func __attribute__((overloadable)) dot(half3 p0, half3 p1);
//half const_func __attribute__((overloadable)) dot(half4 p0, half4 p1);

/**
 * Returns the distance between p0 and p1. This is
 * calculated as length(p0 - p1).
 */
float const_func __attribute__((overloadable)) distance(float p0, float p1);
float const_func __attribute__((overloadable)) distance(float2 p0, float2 p1);
float const_func __attribute__((overloadable)) distance(float3 p0, float3 p1);
float const_func __attribute__((overloadable)) distance(float4 p0, float4 p1);
double const_func __attribute__((overloadable)) distance(double p0, double p1);
double const_func __attribute__((overloadable)) distance(double2 p0, double2 p1);
double const_func __attribute__((overloadable)) distance(double3 p0, double3 p1);
double const_func __attribute__((overloadable)) distance(double4 p0, double4 p1);
//half const_func __attribute__((overloadable)) distance(half p0, half p1);
//half const_func __attribute__((overloadable)) distance(half2 p0, half2 p1);
//half const_func __attribute__((overloadable)) distance(half3 p0, half3 p1);
//half const_func __attribute__((overloadable)) distance(half4 p0, half4 p1);

/**
 * Return the length of vector p, i.e.,
 * sqrt(p.x2 + p.y 2 + ...)
 */
float const_func __attribute__((overloadable)) length(float p);
float const_func __attribute__((overloadable)) length(float2 p);
float const_func __attribute__((overloadable)) length(float3 p);
float const_func __attribute__((overloadable)) length(float4 p);
double const_func __attribute__((overloadable)) length(double p);
double const_func __attribute__((overloadable)) length(double2 p);
double const_func __attribute__((overloadable)) length(double3 p);
double const_func __attribute__((overloadable)) length(double4 p);
//half const_func __attribute__((overloadable)) length(half p);
//half const_func __attribute__((overloadable)) length(half2 p);
//half const_func __attribute__((overloadable)) length(half3 p);
//half const_func __attribute__((overloadable)) length(half4 p);

/**
 * Returns a vector in the same direction as p but with a
 * length of 1.
 */
float const_func __attribute__((overloadable)) normalize(float p);
float2 const_func __attribute__((overloadable)) normalize(float2 p);
float3 const_func __attribute__((overloadable)) normalize(float3 p);
float4 const_func __attribute__((overloadable)) normalize(float4 p);
double const_func __attribute__((overloadable)) normalize(double p);
double2 const_func __attribute__((overloadable)) normalize(double2 p);
double3 const_func __attribute__((overloadable)) normalize(double3 p);
double4 const_func __attribute__((overloadable)) normalize(double4 p);
//half const_func __attribute__((overloadable)) normalize(half p);
//half2 const_func __attribute__((overloadable)) normalize(half2 p);
```

```
//half3 const_func __attribute__((overloadable)) normalize(half3 p);
//half4 const_func __attribute__((overloadable)) normalize(half4 p);

/**
 * Returns fast_length(p0 - p1).
 */
float const_func __attribute__((overloadable)) fast_distance(float p0, float p1);
float const_func __attribute__((overloadable)) fast_distance(float2 p0, float2 p1);
float const_func __attribute__((overloadable)) fast_distance(float3 p0, float3 p1);
float const_func __attribute__((overloadable)) fast_distance(float4 p0, float4 p1);

/**
 * Returns the length of vector p computed as:
 * half_sqrt(p.x2 + p.y2 + ...)
 */
float const_func __attribute__((overloadable)) fast_length(float p);
float const_func __attribute__((overloadable)) fast_length(float2 p);
float const_func __attribute__((overloadable)) fast_length(float3 p);
float const_func __attribute__((overloadable)) fast_length(float4 p);

/**
 * Returns a vector in the same direction as p but with a
 * length of 1. fast_normalize is computed as:
 * p * half_rsqrt (p.x^2 + p.y^2 + ... )
 * The result shall be within 8192 ulps error from the
 * infinitely precise result of
 * if (all(p == 0.0f))
 * result = p;
 * else
 * result = p / sqrt (p.x^2 + p.y^2 + ... );
 * with the following exceptions:
 * 1) If the sum of squares is greater than FLT_MAX
 * then the value of the floating-point values in the
 * result vector are undefined.
 * 2) If the sum of squares is less than FLT_MIN then
 * the implementation may return back p.
 * 3) If the device is in "denorms are flushed to zero"
 * mode, individual operand elements with magnitude
 * less than sqrt(FLT_MIN) may be flushed to zero
 * before proceeding with the calculation.
 */
float const_func __attribute__((overloadable)) fast_normalize(float p);
float2 const_func __attribute__((overloadable)) fast_normalize(float2 p);
float3 const_func __attribute__((overloadable)) fast_normalize(float3 p);
float4 const_func __attribute__((overloadable)) fast_normalize(float4 p);

// Relational functions:

/**
 * intn isequal (floatn x, floatn y)
 * Returns the component-wise compare of x == y.
 */
int const_func __attribute__((overloadable)) isequal(float x, float y);
int2 const_func __attribute__((overloadable)) isequal(float2 x, float2 y);
int3 const_func __attribute__((overloadable)) isequal(float3 x, float3 y);
int4 const_func __attribute__((overloadable)) isequal(float4 x, float4 y);
int8 const_func __attribute__((overloadable)) isequal(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) isequal(float16 x, float16 y);
int const_func __attribute__((overloadable)) isequal(double x, double y);
long2 const_func __attribute__((overloadable)) isequal(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) isequal(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) isequal(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) isequal(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) isequal(double16
```

```
x, double16 y);
//int const_func __attribute__((overloadable)) isequal(half x, half y);
//short2 const_func __attribute__((overloadable)) isequal(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isequal(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isequal(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isequal(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isequal(half16 x, half16 y);

/**
 * Returns the component-wise compare of x != y.
 */
int const_func __attribute__((overloadable)) isnotequal(float
x, float y);
int2 const_func __attribute__((overloadable)) isnotequal(float2
x, float2 y);
int3 const_func __attribute__((overloadable)) isnotequal(float3
x, float3 y);
int4 const_func __attribute__((overloadable)) isnotequal(float4
x, float4 y);
int8 const_func __attribute__((overloadable)) isnotequal(float8
x, float8 y);
int16 const_func __attribute__((overloadable)) isnotequal(float16
x, float16 y);
int const_func __attribute__((overloadable)) isnotequal(double
x, double y);
long2 const_func __attribute__((overloadable)) isnotequal(double2
x, double2 y);
long3 const_func __attribute__((overloadable)) isnotequal(double3
x, double3 y);
long4 const_func __attribute__((overloadable)) isnotequal(double4
x, double4 y);
long8 const_func __attribute__((overloadable)) isnotequal(double8
x, double8 y);
long16 const_func __attribute__((overloadable)) isnotequal(double16
x, double16 y);
//int const_func __attribute__((overloadable)) isnotequal(half x, half y);
//short2 const_func __attribute__((overloadable)) isnotequal(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isnotequal(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isnotequal(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isnotequal(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isnotequal(half16 x, half16 y);

/**
 * Returns the component-wise compare of x > y.
 */
int const_func __attribute__((overloadable)) isgreater(float
x, float y);
int2 const_func __attribute__((overloadable)) isgreater(float2
x, float2 y);
int3 const_func __attribute__((overloadable)) isgreater(float3
x, float3 y);
int4 const_func __attribute__((overloadable)) isgreater(float4
x, float4 y);
int8 const_func __attribute__((overloadable)) isgreater(float8
x, float8 y);
int16 const_func __attribute__((overloadable)) isgreater(float16
x, float16 y);
int const_func __attribute__((overloadable)) isgreater(double
x, double y);
long2 const_func __attribute__((overloadable)) isgreater(double2
x, double2 y);
long3 const_func __attribute__((overloadable)) isgreater(double3
x, double3 y);
long4 const_func __attribute__((overloadable)) isgreater(double4
x, double4 y);
long8 const_func __attribute__((overloadable)) isgreater(double8
x, double8 y);
long16 const_func __attribute__((overloadable)) isgreater(double16
x, double16 y);
//int const_func __attribute__((overloadable)) isgreater(half x, half y);
//short2 const_func __attribute__((overloadable)) isgreater(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isgreater(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isgreater(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isgreater(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isgreater(half16 x, half16 y);

/**
 * Returns the component-wise compare of x >= y.
 */
int const_func __attribute__((overloadable)) isgreaterequal(float
x, float y);
int2 const_func __attribute__((overloadable)) isgreaterequal(float2
x, float2 y);
int3 const_func __attribute__((overloadable)) isgreaterequal(float3
x, float3 y);
int4 const_func __attribute__((overloadable)) isgreaterequal(float4
x, float4 y);
```

```
int8 const_func __attribute__((overloadable)) isgreaterequal(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) isgreaterequal(float16 x, float16 y);
int const_func __attribute__((overloadable)) isgreaterequal(double x, double y);
long2 const_func __attribute__((overloadable)) isgreaterequal(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) isgreaterequal(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) isgreaterequal(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) isgreaterequal(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) isgreaterequal(double16 x, double16 y);
//int const_func __attribute__((overloadable)) isgreaterequal(half x, half y);
//short2 const_func __attribute__((overloadable)) isgreaterequal(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isgreaterequal(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isgreaterequal(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isgreaterequal(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isgreaterequal(half16 x, half16 y);

/**
 * Returns the component-wise compare of x < y.
 */
int const_func __attribute__((overloadable)) isless(float x, float y);
int2 const_func __attribute__((overloadable)) isless(float2 x, float2 y);
int3 const_func __attribute__((overloadable)) isless(float3 x, float3 y);
int4 const_func __attribute__((overloadable)) isless(float4 x, float4 y);
int8 const_func __attribute__((overloadable)) isless(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) isless(float16 x, float16 y);
int const_func __attribute__((overloadable)) isless(double x, double y);
long2 const_func __attribute__((overloadable)) isless(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) isless(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) isless(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) isless(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) isless(double16 x, double16 y);
//int const_func __attribute__((overloadable)) isless(half x, half y);
//short2 const_func __attribute__((overloadable)) isless(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isless(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isless(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isless(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isless(half16 x, half16 y);

/**
 * Returns the component-wise compare of x <= y.
 */
int const_func __attribute__((overloadable)) islessequal(float x, float y);
int2 const_func __attribute__((overloadable)) islessequal(float2 x, float2 y);
int3 const_func __attribute__((overloadable)) islessequal(float3 x, float3 y);
int4 const_func __attribute__((overloadable)) islessequal(float4 x, float4 y);
int8 const_func __attribute__((overloadable)) islessequal(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) islessequal(float16 x, float16 y);
int const_func __attribute__((overloadable)) islessequal(double x, double y);
long2 const_func __attribute__((overloadable)) islessequal(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) islessequal(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) islessequal(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) islessequal(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) islessequal(double16 x, double16 y);
//int const_func __attribute__((overloadable)) islessequal(half x, half y);
//short2 const_func __attribute__((overloadable)) islessequal(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) islessequal(half3 x, half3 y);
```

```
//short4 const_func __attribute__((overloadable)) islessequal(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) islessequal(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) islessequal(half16 x, half16 y);

/**
 * Returns the component-wise compare of
 * (x < y) || (x > y) .
 */
int const_func __attribute__((overloadable)) islessgreater(float x, float y);
int2 const_func __attribute__((overloadable)) islessgreater(float2 x, float2 y);
int3 const_func __attribute__((overloadable)) islessgreater(float3 x, float3 y);
int4 const_func __attribute__((overloadable)) islessgreater(float4 x, float4 y);
int8 const_func __attribute__((overloadable)) islessgreater(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) islessgreater(float16 x, float16 y);
int const_func __attribute__((overloadable)) islessgreater(double x, double y);
long2 const_func __attribute__((overloadable)) islessgreater(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) islessgreater(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) islessgreater(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) islessgreater(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) islessgreater(double16 x, double16 y);
//int const_func __attribute__((overloadable)) islessgreater(half x, half y);
//short2 const_func __attribute__((overloadable)) islessgreater(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) islessgreater(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) islessgreater(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) islessgreater(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) islessgreater(half16 x, half16 y);

/**
 * Test for finite value.
 */
int const_func __attribute__((overloadable)) isfinite(float);
int2 const_func __attribute__((overloadable)) isfinite(float2);
int3 const_func __attribute__((overloadable)) isfinite(float3);
int4 const_func __attribute__((overloadable)) isfinite(float4);
int8 const_func __attribute__((overloadable)) isfinite(float8);
int16 const_func __attribute__((overloadable)) isfinite(float16);
int const_func __attribute__((overloadable)) isfinite(double);
long2 const_func __attribute__((overloadable)) isfinite(double2);
long3 const_func __attribute__((overloadable)) isfinite(double3);
long4 const_func __attribute__((overloadable)) isfinite(double4);
long8 const_func __attribute__((overloadable)) isfinite(double8);
long16 const_func __attribute__((overloadable)) isfinite(double16);
//int const_func __attribute__((overloadable)) isfinite(half);
//short2 const_func __attribute__((overloadable)) isfinite(half2);
//short3 const_func __attribute__((overloadable)) isfinite(half3);
//short4 const_func __attribute__((overloadable)) isfinite(half4);
//short8 const_func __attribute__((overloadable)) isfinite(half8);
//short16 const_func __attribute__((overloadable)) isfinite(half16);

/**
 * Test for infinity value (+ve or -ve) .
 */
int const_func __attribute__((overloadable)) isinf(float);
int2 const_func __attribute__((overloadable)) isinf(float2);
int3 const_func __attribute__((overloadable)) isinf(float3);
int4 const_func __attribute__((overloadable)) isinf(float4);
int8 const_func __attribute__((overloadable)) isinf(float8);
int16 const_func __attribute__((overloadable)) isinf(float16);
int const_func __attribute__((overloadable)) isinf(double);
long2 const_func __attribute__((overloadable)) isinf(double2);
long3 const_func __attribute__((overloadable)) isinf(double3);
long4 const_func __attribute__((overloadable)) isinf(double4);
long8 const_func __attribute__((overloadable)) isinf(double8);
long16 const_func __attribute__((overloadable)) isinf(double16);
//int const_func __attribute__((overloadable)) isinf(half);
//short2 const_func __attribute__((overloadable)) isinf(half2);
//short3 const_func __attribute__((overloadable)) isinf(half3);
//short4 const_func __attribute__((overloadable)) isinf(half4);
//short8 const_func __attribute__((overloadable)) isinf(half8);
//short16 const_func __attribute__((overloadable)) isinf(half16);

/**
 * Test for a NaN.
 */
int const_func __attribute__((overloadable)) isnan(float);
```

```
int2 const_func __attribute__((overloadable)) isnan(float2);
int3 const_func __attribute__((overloadable)) isnan(float3);
int4 const_func __attribute__((overloadable)) isnan(float4);
int8 const_func __attribute__((overloadable)) isnan(float8);
int16 const_func __attribute__((overloadable)) isnan(float16);
int const_func __attribute__((overloadable)) isnan(double);
long2 const_func __attribute__((overloadable)) isnan(double2);
long3 const_func __attribute__((overloadable)) isnan(double3);
long4 const_func __attribute__((overloadable)) isnan(double4);
long8 const_func __attribute__((overloadable)) isnan(double8);
long16 const_func __attribute__((overloadable)) isnan(double16);
//int const_func __attribute__((overloadable)) isnan(half);
//short2 const_func __attribute__((overloadable)) isnan(half2);
//short3 const_func __attribute__((overloadable)) isnan(half3);
//short4 const_func __attribute__((overloadable)) isnan(half4);
//short8 const_func __attribute__((overloadable)) isnan(half8);
//short16 const_func __attribute__((overloadable)) isnan(half16);

/**
 * Test for a normal value.
 */
int const_func __attribute__((overloadable)) isnormal(float);
int2 const_func __attribute__((overloadable)) isnormal(float2);
int3 const_func __attribute__((overloadable)) isnormal(float3);
int4 const_func __attribute__((overloadable)) isnormal(float4);
int8 const_func __attribute__((overloadable)) isnormal(float8);
int16 const_func __attribute__((overloadable)) isnormal(float16);
int const_func __attribute__((overloadable)) isnormal(double);
long2 const_func __attribute__((overloadable)) isnormal(double2);
long3 const_func __attribute__((overloadable)) isnormal(double3);
long4 const_func __attribute__((overloadable)) isnormal(double4);
long8 const_func __attribute__((overloadable)) isnormal(double8);
long16 const_func __attribute__((overloadable)) isnormal(double16);
//int const_func __attribute__((overloadable)) isnormal(half);
//short2 const_func __attribute__((overloadable)) isnormal(half2);
//short3 const_func __attribute__((overloadable)) isnormal(half3);
//short4 const_func __attribute__((overloadable)) isnormal(half4);
//short8 const_func __attribute__((overloadable)) isnormal(half8);
//short16 const_func __attribute__((overloadable)) isnormal(half16);

/**
 * Test if arguments are ordered. isordered() takes
 * arguments x and y, and returns the result
 * isequal(x, x) && isequal(y, y).
 */
int const_func __attribute__((overloadable)) isordered(float x, float y);
int2 const_func __attribute__((overloadable)) isordered(float2 x, float2 y);
int3 const_func __attribute__((overloadable)) isordered(float3 x, float3 y);
int4 const_func __attribute__((overloadable)) isordered(float4 x, float4 y);
int8 const_func __attribute__((overloadable)) isordered(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) isordered(float16 x, float16 y);
int const_func __attribute__((overloadable)) isordered(double x, double y);
long2 const_func __attribute__((overloadable)) isordered(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) isordered(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) isordered(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) isordered(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) isordered(double16 x, double16 y);
//int const_func __attribute__((overloadable)) isordered(half x, half y);
//short2 const_func __attribute__((overloadable)) isordered(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isordered(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isordered(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isordered(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isordered(half16 x, half16 y);

/**
 * Test if arguments are unordered. isunordered()
 * takes arguments x and y, returning non-zero if x or y
 * is NaN, and zero otherwise.
 */
int const_func __attribute__((overloadable)) isunordered(float x, float y);
int2 const_func __attribute__((overloadable)) isunordered(float2 x, float2 y);
int3 const_func __attribute__((overloadable)) isunordered(float3 x, float3 y);
```

```
int4 const_func __attribute__((overloadable)) isunordered(float4 x, float4 y);
int8 const_func __attribute__((overloadable)) isunordered(float8 x, float8 y);
int16 const_func __attribute__((overloadable)) isunordered(float16 x, float16 y);
int const_func __attribute__((overloadable)) isunordered(double x, double y);
long2 const_func __attribute__((overloadable)) isunordered(double2 x, double2 y);
long3 const_func __attribute__((overloadable)) isunordered(double3 x, double3 y);
long4 const_func __attribute__((overloadable)) isunordered(double4 x, double4 y);
long8 const_func __attribute__((overloadable)) isunordered(double8 x, double8 y);
long16 const_func __attribute__((overloadable)) isunordered(double16 x, double16 y);
//int const_func __attribute__((overloadable)) isunordered(half x, half y);
//short2 const_func __attribute__((overloadable)) isunordered(half2 x, half2 y);
//short3 const_func __attribute__((overloadable)) isunordered(half3 x, half3 y);
//short4 const_func __attribute__((overloadable)) isunordered(half4 x, half4 y);
//short8 const_func __attribute__((overloadable)) isunordered(half8 x, half8 y);
//short16 const_func __attribute__((overloadable)) isunordered(half16 x, half16 y);

/**
 * Test for sign bit. The scalar version of the function
 * returns a 1 if the sign bit in the float is set else returns
 * 0. The vector version of the function returns the
 * following for each component in floatn: a -1 if the
 * sign bit in the float is set else returns 0.
 */
int const_func __attribute__((overloadable)) signbit(float);
int2 const_func __attribute__((overloadable)) signbit(float2);
int3 const_func __attribute__((overloadable)) signbit(float3);
int4 const_func __attribute__((overloadable)) signbit(float4);
int8 const_func __attribute__((overloadable)) signbit(float8);
int16 const_func __attribute__((overloadable)) signbit(float16);
int const_func __attribute__((overloadable)) signbit(double);
long2 const_func __attribute__((overloadable)) signbit(double2);
long3 const_func __attribute__((overloadable)) signbit(double3);
long4 const_func __attribute__((overloadable)) signbit(double4);
long8 const_func __attribute__((overloadable)) signbit(double8);
long16 const_func __attribute__((overloadable)) signbit(double16);
//int const_func __attribute__((overloadable)) signbit(half);
//short2 const_func __attribute__((overloadable)) signbit(half2);
//short3 const_func __attribute__((overloadable)) signbit(half3);
//short4 const_func __attribute__((overloadable)) signbit(half4);
//short8 const_func __attribute__((overloadable)) signbit(half8);
//short16 const_func __attribute__((overloadable)) signbit(half16);

/**
 * Returns 1 if the most significant bit in any component
 * of x is set; otherwise returns 0.
 */
int const_func __attribute__((overloadable)) any(char x);
int const_func __attribute__((overloadable)) any(char2 x);
int const_func __attribute__((overloadable)) any(char3 x);
int const_func __attribute__((overloadable)) any(char4 x);
int const_func __attribute__((overloadable)) any(char8 x);
int const_func __attribute__((overloadable)) any(char16 x);
int const_func __attribute__((overloadable)) any(short x);
int const_func __attribute__((overloadable)) any(short2 x);
int const_func __attribute__((overloadable)) any(short3 x);
int const_func __attribute__((overloadable)) any(short4 x);
int const_func __attribute__((overloadable)) any(short8 x);
int const_func __attribute__((overloadable)) any(short16 x);
int const_func __attribute__((overloadable)) any(int x);
int const_func __attribute__((overloadable)) any(int2 x);
int const_func __attribute__((overloadable)) any(int3 x);
int const_func __attribute__((overloadable)) any(int4 x);
int const_func __attribute__((overloadable)) any(int8 x);
int const_func __attribute__((overloadable)) any(int16 x);
int const_func __attribute__((overloadable)) any(long x);
int const_func __attribute__((overloadable)) any(long2 x);
int const_func __attribute__((overloadable)) any(long3 x);
int const_func __attribute__((overloadable)) any(long4 x);
int const_func __attribute__((overloadable)) any(long8 x);
int const_func __attribute__((overloadable)) any(long16 x);

/**
 * Returns 1 if the most significant bit in all components
 * of x is set; otherwise returns 0.
 */
int const_func __attribute__((overloadable)) all(char x);
int const_func __attribute__((overloadable)) all(char2 x);
```

```
int const_func __attribute__((overloadable)) all(char3 x);
int const_func __attribute__((overloadable)) all(char4 x);
int const_func __attribute__((overloadable)) all(char8 x);
int const_func __attribute__((overloadable)) all(char16 x);
int const_func __attribute__((overloadable)) all(short x);
int const_func __attribute__((overloadable)) all(short2 x);
int const_func __attribute__((overloadable)) all(short3 x);
int const_func __attribute__((overloadable)) all(short4 x);
int const_func __attribute__((overloadable)) all(short8 x);
int const_func __attribute__((overloadable)) all(short16
x);
int const_func __attribute__((overloadable)) all(int x);
int const_func __attribute__((overloadable)) all(int2 x);
int const_func __attribute__((overloadable)) all(int3 x);
int const_func __attribute__((overloadable)) all(int4 x);
int const_func __attribute__((overloadable)) all(int8 x);
int const_func __attribute__((overloadable)) all(int16 x);
int const_func __attribute__((overloadable)) all(long x);
int const_func __attribute__((overloadable)) all(long2 x);
int const_func __attribute__((overloadable)) all(long3 x);
int const_func __attribute__((overloadable)) all(long4 x);
int const_func __attribute__((overloadable)) all(long8 x);
int const_func __attribute__((overloadable)) all(long16 x);

/**
 * Each bit of the result is the corresponding bit of a if
 * the corresponding bit of c is 0. Otherwise it is the
 * corresponding bit of b.
 */
char const_func __attribute__((overloadable)) bitselect(char
a, char b, char c);
uchar const_func __attribute__((overloadable)) bitselect(
uchar a, uchar b, uchar c);
char2 const_func __attribute__((overloadable)) bitselect(char2
a, char2 b, char2 c);
uchar2 const_func __attribute__((overloadable)) bitselect(uchar2
a, uchar2 b, uchar2 c);
char3 const_func __attribute__((overloadable)) bitselect(char3
a, char3 b, char3 c);
uchar3 const_func __attribute__((overloadable)) bitselect(uchar3
a, uchar3 b, uchar3 c);
char4 const_func __attribute__((overloadable)) bitselect(char4
a, char4 b, char4 c);
uchar4 const_func __attribute__((overloadable)) bitselect(uchar4
a, uchar4 b, uchar4 c);
char8 const_func __attribute__((overloadable)) bitselect(char8
a, char8 b, char8 c);
uchar8 const_func __attribute__((overloadable)) bitselect(uchar8
a, uchar8 b, uchar8 c);
char16 const_func __attribute__((overloadable)) bitselect(char16
a, char16 b, char16 c);
uchar16 const_func __attribute__((overloadable)) bitselect(uchar16
a, uchar16 b, uchar16 c);
short const_func __attribute__((overloadable)) bitselect(short
a, short b, short c);
ushort const_func __attribute__((overloadable)) bitselect(
ushort a, ushort b, ushort c);
short2 const_func __attribute__((overloadable)) bitselect(short2
a, short2 b, short2 c);
ushort2 const_func __attribute__((overloadable)) bitselect(ushort2
a, ushort2 b, ushort2 c);
short3 const_func __attribute__((overloadable)) bitselect(short3
a, short3 b, short3 c);
ushort3 const_func __attribute__((overloadable)) bitselect(ushort3
a, ushort3 b, ushort3 c);
short4 const_func __attribute__((overloadable)) bitselect(short4
a, short4 b, short4 c);
ushort4 const_func __attribute__((overloadable)) bitselect(ushort4
a, ushort4 b, ushort4 c);
short8 const_func __attribute__((overloadable)) bitselect(short8
a, short8 b, short8 c);
ushort8 const_func __attribute__((overloadable)) bitselect(ushort8
a, ushort8 b, ushort8 c);
short16 const_func __attribute__((overloadable)) bitselect(short16
a, short16 b, short16 c);
ushort16 const_func __attribute__((overloadable)) bitselect(ushort16
a, ushort16 b, ushort16 c);
int const_func __attribute__((overloadable)) bitselect(int
a, int b, int c);
uint const_func __attribute__((overloadable)) bitselect(
uint a, uint b, uint c);
int2 const_func __attribute__((overloadable)) bitselect(int2
a, int2 b, int2 c);
uint2 const_func __attribute__((overloadable)) bitselect(uint2
a, uint2 b, uint2 c);
int3 const_func __attribute__((overloadable)) bitselect(int3
a, int3 b, int3 c);
```

```
uint3 const_func __attribute__((overloadable)) bitselect(uint3 a, uint3 b, uint3 c);
int4 const_func __attribute__((overloadable)) bitselect(int4 a, int4 b, int4 c);
uint4 const_func __attribute__((overloadable)) bitselect(uint4 a, uint4 b, uint4 c);
int8 const_func __attribute__((overloadable)) bitselect(int8 a, int8 b, int8 c);
uint8 const_func __attribute__((overloadable)) bitselect(uint8 a, uint8 b, uint8 c);
int16 const_func __attribute__((overloadable)) bitselect(int16 a, int16 b, int16 c);
uint16 const_func __attribute__((overloadable)) bitselect(uint16 a, uint16 b, uint16 c);
long const_func __attribute__((overloadable)) bitselect(long a, long b, long c);
ulong const_func __attribute__((overloadable)) bitselect(ulong a, ulong b, ulong c);
long2 const_func __attribute__((overloadable)) bitselect(long2 a, long2 b, long2 c);
ulong2 const_func __attribute__((overloadable)) bitselect(ulong2 a, ulong2 b, ulong2 c);
long3 const_func __attribute__((overloadable)) bitselect(long3 a, long3 b, long3 c);
ulong3 const_func __attribute__((overloadable)) bitselect(ulong3 a, ulong3 b, ulong3 c);
long4 const_func __attribute__((overloadable)) bitselect(long4 a, long4 b, long4 c);
ulong4 const_func __attribute__((overloadable)) bitselect(ulong4 a, ulong4 b, ulong4 c);
long8 const_func __attribute__((overloadable)) bitselect(long8 a, long8 b, long8 c);
ulong8 const_func __attribute__((overloadable)) bitselect(ulong8 a, ulong8 b, ulong8 c);
long16 const_func __attribute__((overloadable)) bitselect(long16 a, long16 b, long16 c);
ulong16 const_func __attribute__((overloadable)) bitselect(ulong16 a, ulong16 b, ulong16 c);
float const_func __attribute__((overloadable)) bitselect(float a, float b, float c);
float2 const_func __attribute__((overloadable)) bitselect(float2 a, float2 b, float2 c);
float3 const_func __attribute__((overloadable)) bitselect(float3 a, float3 b, float3 c);
float4 const_func __attribute__((overloadable)) bitselect(float4 a, float4 b, float4 c);
float8 const_func __attribute__((overloadable)) bitselect(float8 a, float8 b, float8 c);
float16 const_func __attribute__((overloadable)) bitselect(float16 a, float16 b, float16 c);
double const_func __attribute__((overloadable)) bitselect(double a, double b, double c);
double2 const_func __attribute__((overloadable)) bitselect(double2 a, double2 b, double2 c);
double3 const_func __attribute__((overloadable)) bitselect(double3 a, double3 b, double3 c);
double4 const_func __attribute__((overloadable)) bitselect(double4 a, double4 b, double4 c);
double8 const_func __attribute__((overloadable)) bitselect(double8 a, double8 b, double8 c);
double16 const_func __attribute__((overloadable)) bitselect(double16 a, double16 b, double16 c);
//half const_func __attribute__((overloadable)) bitselect(half a, half b, half c);
//half2 const_func __attribute__((overloadable)) bitselect(half2 a, half2 b, half2 c);
//half3 const_func __attribute__((overloadable)) bitselect(half3 a, half3 b, half3 c);
//half4 const_func __attribute__((overloadable)) bitselect(half4 a, half4 b, half4 c);
//half8 const_func __attribute__((overloadable)) bitselect(half8 a, half8 b, half8 c);
//half16 const_func __attribute__((overloadable)) bitselect(half16 a, half16 b, half16 c);

/**
 * For each component of a vector type,
 * result[i] = if MSB of c[i] is set ? b[i] : a[i].
 * For a scalar type, result = c ? b : a.
 */
char const_func __attribute__((overloadable)) select(char a, char b, char c);
uchar const_func __attribute__((overloadable)) select(uchar a, uchar b, char c);
char2 const_func __attribute__((overloadable)) select(char2 a, char2 b, char2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2 a, uchar2 b, char2 c);
char3 const_func __attribute__((overloadable)) select(char3 a, char3 b, char3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3 a, uchar3 b, char3 c);
char4 const_func __attribute__((overloadable)) select(char4
```

```
a, char4 b, char4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4
a, uchar4 b, char4 c);
char8 const_func __attribute__((overloadable)) select(char8
a, char8 b, char8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8
a, uchar8 b, char8 c);
char16 const_func __attribute__((overloadable)) select(char16
a, char16 b, char16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16
a, uchar16 b, char16 c);
short const_func __attribute__((overloadable)) select(short
a, short b, char c);
ushort const_func __attribute__((overloadable)) select(
ushort a, ushort b, char c);
short2 const_func __attribute__((overloadable)) select(short2
a, short2 b, char2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2
a, ushort2 b, char2 c);
short3 const_func __attribute__((overloadable)) select(short3
a, short3 b, char3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3
a, ushort3 b, char3 c);
short4 const_func __attribute__((overloadable)) select(short4
a, short4 b, char4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4
a, ushort4 b, char4 c);
short8 const_func __attribute__((overloadable)) select(short8
a, short8 b, char8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8
a, ushort8 b, char8 c);
short16 const_func __attribute__((overloadable)) select(short16
a, short16 b, char16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16
a, ushort16 b, char16 c);
int const_func __attribute__((overloadable)) select(int a, int
b, char c);
uint const_func __attribute__((overloadable)) select(
uint a, uint b, char c);
int2 const_func __attribute__((overloadable)) select(int2
a, int2 b, char2 c);
uint2 const_func __attribute__((overloadable)) select(uint2
a, uint2 b, char2 c);
int3 const_func __attribute__((overloadable)) select(int3
a, int3 b, char3 c);
uint3 const_func __attribute__((overloadable)) select(uint3
a, uint3 b, char3 c);
int4 const_func __attribute__((overloadable)) select(int4
a, int4 b, char4 c);
uint4 const_func __attribute__((overloadable)) select(uint4
a, uint4 b, char4 c);
int8 const_func __attribute__((overloadable)) select(int8
a, int8 b, char8 c);
uint8 const_func __attribute__((overloadable)) select(uint8
a, uint8 b, char8 c);
int16 const_func __attribute__((overloadable)) select(int16
a, int16 b, char16 c);
uint16 const_func __attribute__((overloadable)) select(uint16
a, uint16 b, char16 c);
long const_func __attribute__((overloadable)) select(long
a, long b, char c);
ulong const_func __attribute__((overloadable)) select(
ulong a, ulong b, char c);
long2 const_func __attribute__((overloadable)) select(long2
a, long2 b, char2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2
a, ulong2 b, char2 c);
long3 const_func __attribute__((overloadable)) select(long3
a, long3 b, char3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3
a, ulong3 b, char3 c);
long4 const_func __attribute__((overloadable)) select(long4
a, long4 b, char4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4
a, ulong4 b, char4 c);
long8 const_func __attribute__((overloadable)) select(long8
a, long8 b, char8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8
a, ulong8 b, char8 c);
long16 const_func __attribute__((overloadable)) select(long16
a, long16 b, char16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16
a, ulong16 b, char16 c);
float const_func __attribute__((overloadable)) select(float
a, float b, char c);
float2 const_func __attribute__((overloadable)) select(float2
a, float2 b, char2 c);
```

```
float3 const_func __attribute__((overloadable)) select(float3 a, float3 b, char3 c);
float4 const_func __attribute__((overloadable)) select(float4 a, float4 b, char4 c);
float8 const_func __attribute__((overloadable)) select(float8 a, float8 b, char8 c);
float16 const_func __attribute__((overloadable)) select(float16 a, float16 b, char16 c);
char const_func __attribute__((overloadable)) select(char a, char b, short c);
uchar const_func __attribute__((overloadable)) select(uchar a, uchar b, short c);
char2 const_func __attribute__((overloadable)) select(char2 a, char2 b, short2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2 a, uchar2 b, short2 c);
char3 const_func __attribute__((overloadable)) select(char3 a, char3 b, short3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3 a, uchar3 b, short3 c);
char4 const_func __attribute__((overloadable)) select(char4 a, char4 b, short4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4 a, uchar4 b, short4 c);
char8 const_func __attribute__((overloadable)) select(char8 a, char8 b, short8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8 a, uchar8 b, short8 c);
char16 const_func __attribute__((overloadable)) select(char16 a, char16 b, short16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16 a, uchar16 b, short16 c);
short const_func __attribute__((overloadable)) select(short a, short b, short c);
ushort const_func __attribute__((overloadable)) select(ushort a, ushort b, short c);
short2 const_func __attribute__((overloadable)) select(short2 a, short2 b, short2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2 a, ushort2 b, short2 c);
short3 const_func __attribute__((overloadable)) select(short3 a, short3 b, short3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3 a, ushort3 b, short3 c);
short4 const_func __attribute__((overloadable)) select(short4 a, short4 b, short4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4 a, ushort4 b, short4 c);
short8 const_func __attribute__((overloadable)) select(short8 a, short8 b, short8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8 a, ushort8 b, short8 c);
short16 const_func __attribute__((overloadable)) select(short16 a, short16 b, short16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16 a, ushort16 b, short16 c);
int const_func __attribute__((overloadable)) select(int a, int b, short c);
uint const_func __attribute__((overloadable)) select(uint a, uint b, short c);
int2 const_func __attribute__((overloadable)) select(int2 a, int2 b, short2 c);
uint2 const_func __attribute__((overloadable)) select(uint2 a, uint2 b, short2 c);
int3 const_func __attribute__((overloadable)) select(int3 a, int3 b, short3 c);
uint3 const_func __attribute__((overloadable)) select(uint3 a, uint3 b, short3 c);
int4 const_func __attribute__((overloadable)) select(int4 a, int4 b, short4 c);
uint4 const_func __attribute__((overloadable)) select(uint4 a, uint4 b, short4 c);
int8 const_func __attribute__((overloadable)) select(int8 a, int8 b, short8 c);
uint8 const_func __attribute__((overloadable)) select(uint8 a, uint8 b, short8 c);
int16 const_func __attribute__((overloadable)) select(int16 a, int16 b, short16 c);
uint16 const_func __attribute__((overloadable)) select(uint16 a, uint16 b, short16 c);
long const_func __attribute__((overloadable)) select(long a, long b, short c);
ulong const_func __attribute__((overloadable)) select(ulong a, ulong b, short c);
long2 const_func __attribute__((overloadable)) select(long2 a, long2 b, short2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2
```

```
a, ulong2 b, short2 c);
long3 const_func __attribute__((overloadable)) select(long3
a, long3 b, short3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3
a, ulong3 b, short3 c);
long4 const_func __attribute__((overloadable)) select(long4
a, long4 b, short4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4
a, ulong4 b, short4 c);
long8 const_func __attribute__((overloadable)) select(long8
a, long8 b, short8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8
a, ulong8 b, short8 c);
long16 const_func __attribute__((overloadable)) select(long16
a, long16 b, short16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16
a, ulong16 b, short16 c);
float const_func __attribute__((overloadable)) select(float
a, float b, short c);
float2 const_func __attribute__((overloadable)) select(float2
a, float2 b, short2 c);
float3 const_func __attribute__((overloadable)) select(float3
a, float3 b, short3 c);
float4 const_func __attribute__((overloadable)) select(float4
a, float4 b, short4 c);
float8 const_func __attribute__((overloadable)) select(float8
a, float8 b, short8 c);
float16 const_func __attribute__((overloadable)) select(float16
a, float16 b, short16 c);
char const_func __attribute__((overloadable)) select(char
a, char b, int c);
uchar const_func __attribute__((overloadable)) select(
uchar a, uchar b, int c);
char2 const_func __attribute__((overloadable)) select(char2
a, char2 b, int2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2
a, uchar2 b, int2 c);
char3 const_func __attribute__((overloadable)) select(char3
a, char3 b, int3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3
a, uchar3 b, int3 c);
char4 const_func __attribute__((overloadable)) select(char4
a, char4 b, int4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4
a, uchar4 b, int4 c);
char8 const_func __attribute__((overloadable)) select(char8
a, char8 b, int8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8
a, uchar8 b, int8 c);
char16 const_func __attribute__((overloadable)) select(char16
a, char16 b, int16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16
a, uchar16 b, int16 c);
short const_func __attribute__((overloadable)) select(short
a, short b, int c);
ushort const_func __attribute__((overloadable)) select(
ushort a, ushort b, int c);
short2 const_func __attribute__((overloadable)) select(short2
a, short2 b, int2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2
a, ushort2 b, int2 c);
short3 const_func __attribute__((overloadable)) select(short3
a, short3 b, int3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3
a, ushort3 b, int3 c);
short4 const_func __attribute__((overloadable)) select(short4
a, short4 b, int4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4
a, ushort4 b, int4 c);
short8 const_func __attribute__((overloadable)) select(short8
a, short8 b, int8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8
a, ushort8 b, int8 c);
short16 const_func __attribute__((overloadable)) select(short16
a, short16 b, int16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16
a, ushort16 b, int16 c);
int const_func __attribute__((overloadable)) select(int a, int
b, int c);
uint const_func __attribute__((overloadable)) select(
uint a, uint b, int c);
int2 const_func __attribute__((overloadable)) select(int2
a, int2 b, int2 c);
uint2 const_func __attribute__((overloadable)) select(uint2
a, uint2 b, int2 c);
int3 const_func __attribute__((overloadable)) select(int3
a, int3 b, int3 c);
```

```
uint3 const_func __attribute__((overloadable)) select(uint3
a, uint3 b, int3 c);
int4 const_func __attribute__((overloadable)) select(int4
a, int4 b, int4 c);
uint4 const_func __attribute__((overloadable)) select(uint4
a, uint4 b, int4 c);
int8 const_func __attribute__((overloadable)) select(int8
a, int8 b, int8 c);
uint8 const_func __attribute__((overloadable)) select(uint8
a, uint8 b, int8 c);
int16 const_func __attribute__((overloadable)) select(int16
a, int16 b, int16 c);
uint16 const_func __attribute__((overloadable)) select(uint16
a, uint16 b, int16 c);
long const_func __attribute__((overloadable)) select(long
a, long b, int c);
ulong const_func __attribute__((overloadable)) select(
ulong a, ulong b, int c);
long2 const_func __attribute__((overloadable)) select(long2
a, long2 b, int2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2
a, ulong2 b, int2 c);
long3 const_func __attribute__((overloadable)) select(long3
a, long3 b, int3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3
a, ulong3 b, int3 c);
long4 const_func __attribute__((overloadable)) select(long4
a, long4 b, int4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4
a, ulong4 b, int4 c);
long8 const_func __attribute__((overloadable)) select(long8
a, long8 b, int8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8
a, ulong8 b, int8 c);
long16 const_func __attribute__((overloadable)) select(long16
a, long16 b, int16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16
a, ulong16 b, int16 c);
float const_func __attribute__((overloadable)) select(float
a, float b, int c);
float2 const_func __attribute__((overloadable)) select(float2
a, float2 b, int2 c);
float3 const_func __attribute__((overloadable)) select(float3
a, float3 b, int3 c);
float4 const_func __attribute__((overloadable)) select(float4
a, float4 b, int4 c);
float8 const_func __attribute__((overloadable)) select(float8
a, float8 b, int8 c);
float16 const_func __attribute__((overloadable)) select(float16
a, float16 b, int16 c);
char const_func __attribute__((overloadable)) select(char
a, char b, long c);
uchar const_func __attribute__((overloadable)) select(
uchar a, uchar b, long c);
char2 const_func __attribute__((overloadable)) select(char2
a, char2 b, long2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2
a, uchar2 b, long2 c);
char3 const_func __attribute__((overloadable)) select(char3
a, char3 b, long3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3
a, uchar3 b, long3 c);
char4 const_func __attribute__((overloadable)) select(char4
a, char4 b, long4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4
a, uchar4 b, long4 c);
char8 const_func __attribute__((overloadable)) select(char8
a, char8 b, long8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8
a, uchar8 b, long8 c);
char16 const_func __attribute__((overloadable)) select(char16
a, char16 b, long16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16
a, uchar16 b, long16 c);
short const_func __attribute__((overloadable)) select(short
a, short b, long c);
ushort const_func __attribute__((overloadable)) select(
ushort a, ushort b, long c);
short2 const_func __attribute__((overloadable)) select(short2
a, short2 b, long2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2
a, ushort2 b, long2 c);
short3 const_func __attribute__((overloadable)) select(short3
a, short3 b, long3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3
a, ushort3 b, long3 c);
short4 const_func __attribute__((overloadable)) select(short4
```

```
a, short4 b, long4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4
a, ushort4 b, long4 c);
short8 const_func __attribute__((overloadable)) select(short8
a, short8 b, long8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8
a, ushort8 b, long8 c);
short16 const_func __attribute__((overloadable)) select(short16
a, short16 b, long16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16
a, ushort16 b, long16 c);
int const_func __attribute__((overloadable)) select(int a, int
b, long c);
uint const_func __attribute__((overloadable)) select(
uint a, uint b, long c);
int2 const_func __attribute__((overloadable)) select(int2
a, int2 b, long2 c);
uint2 const_func __attribute__((overloadable)) select(uint2
a, uint2 b, long2 c);
int3 const_func __attribute__((overloadable)) select(int3
a, int3 b, long3 c);
uint3 const_func __attribute__((overloadable)) select(uint3
a, uint3 b, long3 c);
int4 const_func __attribute__((overloadable)) select(int4
a, int4 b, long4 c);
uint4 const_func __attribute__((overloadable)) select(uint4
a, uint4 b, long4 c);
int8 const_func __attribute__((overloadable)) select(int8
a, int8 b, long8 c);
uint8 const_func __attribute__((overloadable)) select(uint8
a, uint8 b, long8 c);
int16 const_func __attribute__((overloadable)) select(int16
a, int16 b, long16 c);
uint16 const_func __attribute__((overloadable)) select(uint16
a, uint16 b, long16 c);
long const_func __attribute__((overloadable)) select(long
a, long b, long c);
ulong const_func __attribute__((overloadable)) select(
ulong a, ulong b, long c);
long2 const_func __attribute__((overloadable)) select(long2
a, long2 b, long2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2
a, ulong2 b, long2 c);
long3 const_func __attribute__((overloadable)) select(long3
a, long3 b, long3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3
a, ulong3 b, long3 c);
long4 const_func __attribute__((overloadable)) select(long4
a, long4 b, long4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4
a, ulong4 b, long4 c);
long8 const_func __attribute__((overloadable)) select(long8
a, long8 b, long8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8
a, ulong8 b, long8 c);
long16 const_func __attribute__((overloadable)) select(long16
a, long16 b, long16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16
a, ulong16 b, long16 c);
float const_func __attribute__((overloadable)) select(float
a, float b, long c);
float2 const_func __attribute__((overloadable)) select(float2
a, float2 b, long2 c);
float3 const_func __attribute__((overloadable)) select(float3
a, float3 b, long3 c);
float4 const_func __attribute__((overloadable)) select(float4
a, float4 b, long4 c);
float8 const_func __attribute__((overloadable)) select(float8
a, float8 b, long8 c);
float16 const_func __attribute__((overloadable)) select(float16
a, float16 b, long16 c);
char const_func __attribute__((overloadable)) select(char
a, char b, uchar c);
uchar const_func __attribute__((overloadable)) select(
uchar a, uchar b, uchar c);
char2 const_func __attribute__((overloadable)) select(char2
a, char2 b, uchar2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2
a, uchar2 b, uchar2 c);
char3 const_func __attribute__((overloadable)) select(char3
a, char3 b, uchar3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3
a, uchar3 b, uchar3 c);
char4 const_func __attribute__((overloadable)) select(char4
a, char4 b, uchar4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4
a, uchar4 b, uchar4 c);
```

```
char8 const_func __attribute__((overloadable)) select(char8 a, char8 b, uchar8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8 a, uchar8 b, uchar8 c);
char16 const_func __attribute__((overloadable)) select(char16 a, char16 b, uchar16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16 a, uchar16 b, uchar16 c);
short const_func __attribute__((overloadable)) select(short a, short b, uchar c);
ushort const_func __attribute__((overloadable)) select(ushort a, ushort b, uchar c);
short2 const_func __attribute__((overloadable)) select(short2 a, short2 b, uchar2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2 a, ushort2 b, uchar2 c);
short3 const_func __attribute__((overloadable)) select(short3 a, short3 b, uchar3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3 a, ushort3 b, uchar3 c);
short4 const_func __attribute__((overloadable)) select(short4 a, short4 b, uchar4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4 a, ushort4 b, uchar4 c);
short8 const_func __attribute__((overloadable)) select(short8 a, short8 b, uchar8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8 a, ushort8 b, uchar8 c);
short16 const_func __attribute__((overloadable)) select(short16 a, short16 b, uchar16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16 a, ushort16 b, uchar16 c);
int const_func __attribute__((overloadable)) select(int a, int b, uchar c);
uint const_func __attribute__((overloadable)) select(uint a, uint b, uchar c);
int2 const_func __attribute__((overloadable)) select(int2 a, int2 b, uchar2 c);
uint2 const_func __attribute__((overloadable)) select(uint2 a, uint2 b, uchar2 c);
int3 const_func __attribute__((overloadable)) select(int3 a, int3 b, uchar3 c);
uint3 const_func __attribute__((overloadable)) select(uint3 a, uint3 b, uchar3 c);
int4 const_func __attribute__((overloadable)) select(int4 a, int4 b, uchar4 c);
uint4 const_func __attribute__((overloadable)) select(uint4 a, uint4 b, uchar4 c);
int8 const_func __attribute__((overloadable)) select(int8 a, int8 b, uchar8 c);
uint8 const_func __attribute__((overloadable)) select(uint8 a, uint8 b, uchar8 c);
int16 const_func __attribute__((overloadable)) select(int16 a, int16 b, uchar16 c);
uint16 const_func __attribute__((overloadable)) select(uint16 a, uint16 b, uchar16 c);
long const_func __attribute__((overloadable)) select(long a, long b, uchar c);
ulong const_func __attribute__((overloadable)) select(ulong a, ulong b, uchar c);
long2 const_func __attribute__((overloadable)) select(long2 a, long2 b, uchar2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2 a, ulong2 b, uchar2 c);
long3 const_func __attribute__((overloadable)) select(long3 a, long3 b, uchar3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3 a, ulong3 b, uchar3 c);
long4 const_func __attribute__((overloadable)) select(long4 a, long4 b, uchar4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4 a, ulong4 b, uchar4 c);
long8 const_func __attribute__((overloadable)) select(long8 a, long8 b, uchar8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8 a, ulong8 b, uchar8 c);
long16 const_func __attribute__((overloadable)) select(long16 a, long16 b, uchar16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16 a, ulong16 b, uchar16 c);
float const_func __attribute__((overloadable)) select(float a, float b, uchar c);
float2 const_func __attribute__((overloadable)) select(float2 a, float2 b, uchar2 c);
float3 const_func __attribute__((overloadable)) select(float3 a, float3 b, uchar3 c);
float4 const_func __attribute__((overloadable)) select(float4
```

```
a, float4 b, uchar4 c);
float8 const_func __attribute__((overloadable)) select(float8
a, float8 b, uchar8 c);
float16 const_func __attribute__((overloadable)) select(float16
a, float16 b, uchar16 c);
char const_func __attribute__((overloadable)) select(char
a, char b, ushort c);
uchar const_func __attribute__((overloadable)) select(
uchar a, uchar b, ushort c);
char2 const_func __attribute__((overloadable)) select(char2
a, char2 b, ushort2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2
a, uchar2 b, ushort2 c);
char3 const_func __attribute__((overloadable)) select(char3
a, char3 b, ushort3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3
a, uchar3 b, ushort3 c);
char4 const_func __attribute__((overloadable)) select(char4
a, char4 b, ushort4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4
a, uchar4 b, ushort4 c);
char8 const_func __attribute__((overloadable)) select(char8
a, char8 b, ushort8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8
a, uchar8 b, ushort8 c);
char16 const_func __attribute__((overloadable)) select(char16
a, char16 b, ushort16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16
a, uchar16 b, ushort16 c);
short const_func __attribute__((overloadable)) select(short
a, short b, ushort c);
ushort const_func __attribute__((overloadable)) select(
ushort a, ushort b, ushort c);
short2 const_func __attribute__((overloadable)) select(short2
a, short2 b, ushort2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2
a, ushort2 b, ushort2 c);
short3 const_func __attribute__((overloadable)) select(short3
a, short3 b, ushort3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3
a, ushort3 b, ushort3 c);
short4 const_func __attribute__((overloadable)) select(short4
a, short4 b, ushort4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4
a, ushort4 b, ushort4 c);
short8 const_func __attribute__((overloadable)) select(short8
a, short8 b, ushort8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8
a, ushort8 b, ushort8 c);
short16 const_func __attribute__((overloadable)) select(short16
a, short16 b, ushort16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16
a, ushort16 b, ushort16 c);
int const_func __attribute__((overloadable)) select(int a, int
b, ushort c);
uint const_func __attribute__((overloadable)) select(
uint a, uint b, ushort c);
int2 const_func __attribute__((overloadable)) select(int2
a, int2 b, ushort2 c);
uint2 const_func __attribute__((overloadable)) select(uint2
a, uint2 b, ushort2 c);
int3 const_func __attribute__((overloadable)) select(int3
a, int3 b, ushort3 c);
uint3 const_func __attribute__((overloadable)) select(uint3
a, uint3 b, ushort3 c);
int4 const_func __attribute__((overloadable)) select(int4
a, int4 b, ushort4 c);
uint4 const_func __attribute__((overloadable)) select(uint4
a, uint4 b, ushort4 c);
int8 const_func __attribute__((overloadable)) select(int8
a, int8 b, ushort8 c);
uint8 const_func __attribute__((overloadable)) select(uint8
a, uint8 b, ushort8 c);
int16 const_func __attribute__((overloadable)) select(int16
a, int16 b, ushort16 c);
uint16 const_func __attribute__((overloadable)) select(uint16
a, uint16 b, ushort16 c);
long const_func __attribute__((overloadable)) select(long
a, long b, ushort c);
ulong const_func __attribute__((overloadable)) select(
ulong a, ulong b, ushort c);
long2 const_func __attribute__((overloadable)) select(long2
a, long2 b, ushort2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2
a, ulong2 b, ushort2 c);
long3 const_func __attribute__((overloadable)) select(long3
a, long3 b, ushort3 c);
```

```
ulong3 const_func __attribute__((overloadable)) select(ulong3 a, ulong3 b, ushort3 c);
long4 const_func __attribute__((overloadable)) select(long4 a, long4 b, ushort4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4 a, ulong4 b, ushort4 c);
long8 const_func __attribute__((overloadable)) select(long8 a, long8 b, ushort8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8 a, ulong8 b, ushort8 c);
long16 const_func __attribute__((overloadable)) select(long16 a, long16 b, ushort16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16 a, ulong16 b, ushort16 c);
float const_func __attribute__((overloadable)) select(float a, float b, ushort c);
float2 const_func __attribute__((overloadable)) select(float2 a, float2 b, ushort2 c);
float3 const_func __attribute__((overloadable)) select(float3 a, float3 b, ushort3 c);
float4 const_func __attribute__((overloadable)) select(float4 a, float4 b, ushort4 c);
float8 const_func __attribute__((overloadable)) select(float8 a, float8 b, ushort8 c);
float16 const_func __attribute__((overloadable)) select(float16 a, float16 b, ushort16 c);
char const_func __attribute__((overloadable)) select(char a, char b, uint c);
uchar const_func __attribute__((overloadable)) select(uchar a, uchar b, uint c);
char2 const_func __attribute__((overloadable)) select(char2 a, char2 b, uint2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2 a, uchar2 b, uint2 c);
char3 const_func __attribute__((overloadable)) select(char3 a, char3 b, uint3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3 a, uchar3 b, uint3 c);
char4 const_func __attribute__((overloadable)) select(char4 a, char4 b, uint4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4 a, uchar4 b, uint4 c);
char8 const_func __attribute__((overloadable)) select(char8 a, char8 b, uint8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8 a, uchar8 b, uint8 c);
char16 const_func __attribute__((overloadable)) select(char16 a, char16 b, uint16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16 a, uchar16 b, uint16 c);
short const_func __attribute__((overloadable)) select(short a, short b, uint c);
ushort const_func __attribute__((overloadable)) select(ushort a, ushort b, uint c);
short2 const_func __attribute__((overloadable)) select(short2 a, short2 b, uint2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2 a, ushort2 b, uint2 c);
short3 const_func __attribute__((overloadable)) select(short3 a, short3 b, uint3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3 a, ushort3 b, uint3 c);
short4 const_func __attribute__((overloadable)) select(short4 a, short4 b, uint4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4 a, ushort4 b, uint4 c);
short8 const_func __attribute__((overloadable)) select(short8 a, short8 b, uint8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8 a, ushort8 b, uint8 c);
short16 const_func __attribute__((overloadable)) select(short16 a, short16 b, uint16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16 a, ushort16 b, uint16 c);
int const_func __attribute__((overloadable)) select(int a, int b, uint c);
uint const_func __attribute__((overloadable)) select(uint a, uint b, uint c);
int2 const_func __attribute__((overloadable)) select(int2 a, int2 b, uint2 c);
uint2 const_func __attribute__((overloadable)) select(uint2 a, uint2 b, uint2 c);
int3 const_func __attribute__((overloadable)) select(int3 a, int3 b, uint3 c);
uint3 const_func __attribute__((overloadable)) select(uint3 a, uint3 b, uint3 c);
int4 const_func __attribute__((overloadable)) select(int4
```

```
a, int4 b, uint4 c);
uint4 const_func __attribute__((overloadable)) select(uint4
a, uint4 b, uint4 c);
int8 const_func __attribute__((overloadable)) select(int8
a, int8 b, uint8 c);
uint8 const_func __attribute__((overloadable)) select(uint8
a, uint8 b, uint8 c);
int16 const_func __attribute__((overloadable)) select(int16
a, int16 b, uint16 c);
uint16 const_func __attribute__((overloadable)) select(uint16
a, uint16 b, uint16 c);
long const_func __attribute__((overloadable)) select(long
a, long b, uint c);
ulong const_func __attribute__((overloadable)) select(
ulong a, ulong b, uint c);
long2 const_func __attribute__((overloadable)) select(long2
a, long2 b, uint2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2
a, ulong2 b, uint2 c);
long3 const_func __attribute__((overloadable)) select(long3
a, long3 b, uint3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3
a, ulong3 b, uint3 c);
long4 const_func __attribute__((overloadable)) select(long4
a, long4 b, uint4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4
a, ulong4 b, uint4 c);
long8 const_func __attribute__((overloadable)) select(long8
a, long8 b, uint8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8
a, ulong8 b, uint8 c);
long16 const_func __attribute__((overloadable)) select(long16
a, long16 b, uint16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16
a, ulong16 b, uint16 c);
float const_func __attribute__((overloadable)) select(float
a, float b, uint c);
float2 const_func __attribute__((overloadable)) select(float2
a, float2 b, uint2 c);
float3 const_func __attribute__((overloadable)) select(float3
a, float3 b, uint3 c);
float4 const_func __attribute__((overloadable)) select(float4
a, float4 b, uint4 c);
float8 const_func __attribute__((overloadable)) select(float8
a, float8 b, uint8 c);
float16 const_func __attribute__((overloadable)) select(float16
a, float16 b, uint16 c);
char const_func __attribute__((overloadable)) select(char
a, char b, ulong c);
uchar const_func __attribute__((overloadable)) select(
uchar a, uchar b, ulong c);
char2 const_func __attribute__((overloadable)) select(char2
a, char2 b, ulong2 c);
uchar2 const_func __attribute__((overloadable)) select(uchar2
a, uchar2 b, ulong2 c);
char3 const_func __attribute__((overloadable)) select(char3
a, char3 b, ulong3 c);
uchar3 const_func __attribute__((overloadable)) select(uchar3
a, uchar3 b, ulong3 c);
char4 const_func __attribute__((overloadable)) select(char4
a, char4 b, ulong4 c);
uchar4 const_func __attribute__((overloadable)) select(uchar4
a, uchar4 b, ulong4 c);
char8 const_func __attribute__((overloadable)) select(char8
a, char8 b, ulong8 c);
uchar8 const_func __attribute__((overloadable)) select(uchar8
a, uchar8 b, ulong8 c);
char16 const_func __attribute__((overloadable)) select(char16
a, char16 b, ulong16 c);
uchar16 const_func __attribute__((overloadable)) select(uchar16
a, uchar16 b, ulong16 c);
short const_func __attribute__((overloadable)) select(short
a, short b, ulong c);
ushort const_func __attribute__((overloadable)) select(
ushort a, ushort b, ulong c);
short2 const_func __attribute__((overloadable)) select(short2
a, short2 b, ulong2 c);
ushort2 const_func __attribute__((overloadable)) select(ushort2
a, ushort2 b, ulong2 c);
short3 const_func __attribute__((overloadable)) select(short3
a, short3 b, ulong3 c);
ushort3 const_func __attribute__((overloadable)) select(ushort3
a, ushort3 b, ulong3 c);
short4 const_func __attribute__((overloadable)) select(short4
a, short4 b, ulong4 c);
ushort4 const_func __attribute__((overloadable)) select(ushort4
a, ushort4 b, ulong4 c);
```

```
short8 const_func __attribute__((overloadable)) select(short8 a, short8 b, ulong8 c);
ushort8 const_func __attribute__((overloadable)) select(ushort8 a, ushort8 b, ulong8 c);
short16 const_func __attribute__((overloadable)) select(short16 a, short16 b, ulong16 c);
ushort16 const_func __attribute__((overloadable)) select(ushort16 a, ushort16 b, ulong16 c);
int const_func __attribute__((overloadable)) select(int a, int b, ulong c);
uint const_func __attribute__((overloadable)) select(uint a, uint b, ulong c);
int2 const_func __attribute__((overloadable)) select(int2 a, int2 b, ulong2 c);
uint2 const_func __attribute__((overloadable)) select(uint2 a, uint2 b, ulong2 c);
int3 const_func __attribute__((overloadable)) select(int3 a, int3 b, ulong3 c);
uint3 const_func __attribute__((overloadable)) select(uint3 a, uint3 b, ulong3 c);
int4 const_func __attribute__((overloadable)) select(int4 a, int4 b, ulong4 c);
uint4 const_func __attribute__((overloadable)) select(uint4 a, uint4 b, ulong4 c);
int8 const_func __attribute__((overloadable)) select(int8 a, int8 b, ulong8 c);
uint8 const_func __attribute__((overloadable)) select(uint8 a, uint8 b, ulong8 c);
int16 const_func __attribute__((overloadable)) select(int16 a, int16 b, ulong16 c);
uint16 const_func __attribute__((overloadable)) select(uint16 a, uint16 b, ulong16 c);
long const_func __attribute__((overloadable)) select(long a, long b, ulong c);
ulong const_func __attribute__((overloadable)) select(ulong a, ulong b, ulong c);
long2 const_func __attribute__((overloadable)) select(long2 a, long2 b, ulong2 c);
ulong2 const_func __attribute__((overloadable)) select(ulong2 a, ulong2 b, ulong2 c);
long3 const_func __attribute__((overloadable)) select(long3 a, long3 b, ulong3 c);
ulong3 const_func __attribute__((overloadable)) select(ulong3 a, ulong3 b, ulong3 c);
long4 const_func __attribute__((overloadable)) select(long4 a, long4 b, ulong4 c);
ulong4 const_func __attribute__((overloadable)) select(ulong4 a, ulong4 b, ulong4 c);
long8 const_func __attribute__((overloadable)) select(long8 a, long8 b, ulong8 c);
ulong8 const_func __attribute__((overloadable)) select(ulong8 a, ulong8 b, ulong8 c);
long16 const_func __attribute__((overloadable)) select(long16 a, long16 b, ulong16 c);
ulong16 const_func __attribute__((overloadable)) select(ulong16 a, ulong16 b, ulong16 c);
float const_func __attribute__((overloadable)) select(float a, float b, ulong c);
float2 const_func __attribute__((overloadable)) select(float2 a, float2 b, ulong2 c);
float3 const_func __attribute__((overloadable)) select(float3 a, float3 b, ulong3 c);
float4 const_func __attribute__((overloadable)) select(float4 a, float4 b, ulong4 c);
float8 const_func __attribute__((overloadable)) select(float8 a, float8 b, ulong8 c);
float16 const_func __attribute__((overloadable)) select(float16 a, float16 b, ulong16 c);
double const_func __attribute__((overloadable)) select(double a, double b, long c);
double2 const_func __attribute__((overloadable)) select(double2 a, double2 b, long2 c);
double3 const_func __attribute__((overloadable)) select(double3 a, double3 b, long3 c);
double4 const_func __attribute__((overloadable)) select(double4 a, double4 b, long4 c);
double8 const_func __attribute__((overloadable)) select(double8 a, double8 b, long8 c);
double16 const_func __attribute__((overloadable)) select(double16 a, double16 b, long16 c);
double const_func __attribute__((overloadable)) select(double a, double b, ulong c);
double2 const_func __attribute__((overloadable)) select(double2 a, double2 b, ulong2 c);
double3 const_func __attribute__((overloadable)) select(double3 a, double3 b, ulong3 c);
double4 const_func __attribute__((overloadable)) select(double4
```

```
a, double4 b, ulong4 c);
double8 const_func __attribute__((overloadable)) select(double8 a, double8 b, ulong8 c);
double16 const_func __attribute__((overloadable)) select(double16 a, double16 b, ulong16 c);
//half const_func __attribute__((overloadable)) select(half a, half b, short c);
//half2 const_func __attribute__((overloadable)) select(half2 a, half2 b, short2 c);
//half3 const_func __attribute__((overloadable)) select(half3 a, half3 b, short3 c);
//half4 const_func __attribute__((overloadable)) select(half4 a, half4 b, short4 c);
//half8 const_func __attribute__((overloadable)) select(half8 a, half8 b, short8 c);
//half16 const_func __attribute__((overloadable)) select(half16 a, half16 b, short16 c);
//half const_func __attribute__((overloadable)) select(half a, half b, ushort c);
//half2 const_func __attribute__((overloadable)) select(half2 a, half2 b, ushort2 c);
//half3 const_func __attribute__((overloadable)) select(half3 a, half3 b, ushort3 c);
//half4 const_func __attribute__((overloadable)) select(half4 a, half4 b, ushort4 c);
//half8 const_func __attribute__((overloadable)) select(half8 a, half8 b, ushort8 c);
//half16 const_func __attribute__((overloadable)) select(half16 a, half16 b, ushort16 c);

// Vector data load and store functions

/**
 * Return sizeof (gentypen) bytes of data read
 * from address (p + (offset * n)). The address
 * computed as (p + (offset * n)) must be 8-bit
 * aligned if gentype is char, uchar; 16-bit
 * aligned if gentype is short, ushort; 32-bit
 * aligned if gentype is int, uint, float; 64-bit
 * aligned if gentype is long, ulong.
 */
char2 __attribute__((overloadable)) vload2(size_t offset, const __global char *p);
uchar2 __attribute__((overloadable)) vload2(size_t offset, const __global uchar *p);
short2 __attribute__((overloadable)) vload2(size_t offset, const __global short *p);
ushort2 __attribute__((overloadable)) vload2(size_t offset, const __global ushort *p);
int2 __attribute__((overloadable)) vload2(size_t offset, const __global int *p);
uint2 __attribute__((overloadable)) vload2(size_t offset, const __global uint *p);
long2 __attribute__((overloadable)) vload2(size_t offset, const __global long *p);
ulong2 __attribute__((overloadable)) vload2(size_t offset, const __global ulong *p);
float2 __attribute__((overloadable)) vload2(size_t offset, const __global float *p);
char3 __attribute__((overloadable)) vload3(size_t offset, const __global char *p);
uchar3 __attribute__((overloadable)) vload3(size_t offset, const __global uchar *p);
short3 __attribute__((overloadable)) vload3(size_t offset, const __global short *p);
ushort3 __attribute__((overloadable)) vload3(size_t offset, const __global ushort *p);
int3 __attribute__((overloadable)) vload3(size_t offset, const __global int *p);
uint3 __attribute__((overloadable)) vload3(size_t offset, const __global uint *p);
long3 __attribute__((overloadable)) vload3(size_t offset, const __global long *p);
ulong3 __attribute__((overloadable)) vload3(size_t offset, const __global ulong *p);
float3 __attribute__((overloadable)) vload3(size_t offset, const __global float *p);
char4 __attribute__((overloadable)) vload4(size_t offset, const __global char *p);
uchar4 __attribute__((overloadable)) vload4(size_t offset, const __global uchar *p);
short4 __attribute__((overloadable)) vload4(size_t offset, const __global short *p);
ushort4 __attribute__((overloadable)) vload4(size_t offset, const __global ushort *p);
int4 __attribute__((overloadable)) vload4(size_t offset, const __global int *p);
uint4 __attribute__((overloadable)) vload4(size_t offset, const __global uint *p);
long4 __attribute__((overloadable)) vload4(size_t offset, const __global long *p);
ulong4 __attribute__((overloadable)) vload4(size_t offset, const __global ulong *p);
float4 __attribute__((overloadable)) vload4(size_t offset, const __global float *p);
char8 __attribute__((overloadable)) vload8(size_t offset, const __global char *p);
uchar8 __attribute__((overloadable)) vload8(size_t offset, const __global uchar *p);
```

```
short8 __attribute__((overloadable)) vload8(size_t offset, const __global short *p);
ushort8 __attribute__((overloadable)) vload8(size_t offset, const __global ushort *p);
int8 __attribute__((overloadable)) vload8(size_t offset, const __global int *p);
uint8 __attribute__((overloadable)) vload8(size_t offset, const __global uint *p);
long8 __attribute__((overloadable)) vload8(size_t offset, const __global long *p);
ulong8 __attribute__((overloadable)) vload8(size_t offset, const __global ulong *p);
float8 __attribute__((overloadable)) vload8(size_t offset, const __global float *p);
char16 __attribute__((overloadable)) vload16(size_t offset, const __global char *p);
uchar16 __attribute__((overloadable)) vload16(size_t offset, const __global uchar *p);
short16 __attribute__((overloadable)) vload16(size_t offset, const __global short *p);
ushort16 __attribute__((overloadable)) vload16(size_t offset, const __global ushort *p);
int16 __attribute__((overloadable)) vload16(size_t offset, const __global int *p);
uint16 __attribute__((overloadable)) vload16(size_t offset, const __global uint *p);
long16 __attribute__((overloadable)) vload16(size_t offset, const __global long *p);
ulong16 __attribute__((overloadable)) vload16(size_t offset, const __global ulong *p);
float16 __attribute__((overloadable)) vload16(size_t offset, const __global float *p);
char2 __attribute__((overloadable)) vload2(size_t offset, const __local char *p);
uchar2 __attribute__((overloadable)) vload2(size_t offset, const __local uchar *p);
short2 __attribute__((overloadable)) vload2(size_t offset, const __local short *p);
ushort2 __attribute__((overloadable)) vload2(size_t offset, const __local ushort *p);
int2 __attribute__((overloadable)) vload2(size_t offset, const __local int *p);
uint2 __attribute__((overloadable)) vload2(size_t offset, const __local uint *p);
long2 __attribute__((overloadable)) vload2(size_t offset, const __local long *p);
ulong2 __attribute__((overloadable)) vload2(size_t offset, const __local ulong *p);
float2 __attribute__((overloadable)) vload2(size_t offset, const __local float *p);
char3 __attribute__((overloadable)) vload3(size_t offset, const __local char *p);
uchar3 __attribute__((overloadable)) vload3(size_t offset, const __local uchar *p);
short3 __attribute__((overloadable)) vload3(size_t offset, const __local short *p);
ushort3 __attribute__((overloadable)) vload3(size_t offset, const __local ushort *p);
int3 __attribute__((overloadable)) vload3(size_t offset, const __local int *p);
uint3 __attribute__((overloadable)) vload3(size_t offset, const __local uint *p);
long3 __attribute__((overloadable)) vload3(size_t offset, const __local long *p);
ulong3 __attribute__((overloadable)) vload3(size_t offset, const __local ulong *p);
float3 __attribute__((overloadable)) vload3(size_t offset, const __local float *p);
char4 __attribute__((overloadable)) vload4(size_t offset, const __local char *p);
uchar4 __attribute__((overloadable)) vload4(size_t offset, const __local uchar *p);
short4 __attribute__((overloadable)) vload4(size_t offset, const __local short *p);
ushort4 __attribute__((overloadable)) vload4(size_t offset, const __local ushort *p);
int4 __attribute__((overloadable)) vload4(size_t offset, const __local int *p);
uint4 __attribute__((overloadable)) vload4(size_t offset, const __local uint *p);
long4 __attribute__((overloadable)) vload4(size_t offset, const __local long *p);
ulong4 __attribute__((overloadable)) vload4(size_t offset, const __local ulong *p);
float4 __attribute__((overloadable)) vload4(size_t offset, const __local float *p);
char8 __attribute__((overloadable)) vload8(size_t offset, const __local char *
```

```
p);
uchar8 __attribute__((overloadable)) vload8(size_t offset, const __local uchar *p);
short8 __attribute__((overloadable)) vload8(size_t offset, const __local short *p);
ushort8 __attribute__((overloadable)) vload8(size_t offset, const __local ushort *p);
int8 __attribute__((overloadable)) vload8(size_t offset, const __local int *p);
uint8 __attribute__((overloadable)) vload8(size_t offset, const __local uint *p);
long8 __attribute__((overloadable)) vload8(size_t offset, const __local long *p);
ulong8 __attribute__((overloadable)) vload8(size_t offset, const __local ulong *p);
float8 __attribute__((overloadable)) vload8(size_t offset, const __local float *p);
char16 __attribute__((overloadable)) vload16(size_t offset, const __local char *p);
uchar16 __attribute__((overloadable)) vload16(size_t offset, const __local uchar *p);
short16 __attribute__((overloadable)) vload16(size_t offset, const __local short *p);
ushort16 __attribute__((overloadable)) vload16(size_t offset, const __local ushort *p);
int16 __attribute__((overloadable)) vload16(size_t offset, const __local int *p);
uint16 __attribute__((overloadable)) vload16(size_t offset, const __local uint *p);
long16 __attribute__((overloadable)) vload16(size_t offset, const __local long *p);
ulong16 __attribute__((overloadable)) vload16(size_t offset, const __local ulong *p);
float16 __attribute__((overloadable)) vload16(size_t offset, const __local float *p);
char2 __attribute__((overloadable)) vload2(size_t offset, const __private char *p);
uchar2 __attribute__((overloadable)) vload2(size_t offset, const __private uchar *p);
short2 __attribute__((overloadable)) vload2(size_t offset, const __private short *p);
ushort2 __attribute__((overloadable)) vload2(size_t offset, const __private ushort *p);
int2 __attribute__((overloadable)) vload2(size_t offset, const __private int *p);
uint2 __attribute__((overloadable)) vload2(size_t offset, const __private uint *p);
long2 __attribute__((overloadable)) vload2(size_t offset, const __private long *p);
ulong2 __attribute__((overloadable)) vload2(size_t offset, const __private ulong *p);
float2 __attribute__((overloadable)) vload2(size_t offset, const __private float *p);
char3 __attribute__((overloadable)) vload3(size_t offset, const __private char *p);
uchar3 __attribute__((overloadable)) vload3(size_t offset, const __private uchar *p);
short3 __attribute__((overloadable)) vload3(size_t offset, const __private short *p);
ushort3 __attribute__((overloadable)) vload3(size_t offset, const __private ushort *p);
int3 __attribute__((overloadable)) vload3(size_t offset, const __private int *p);
uint3 __attribute__((overloadable)) vload3(size_t offset, const __private uint *p);
long3 __attribute__((overloadable)) vload3(size_t offset, const __private long *p);
ulong3 __attribute__((overloadable)) vload3(size_t offset, const __private ulong *p);
float3 __attribute__((overloadable)) vload3(size_t offset, const __private float *p);
char4 __attribute__((overloadable)) vload4(size_t offset, const __private char *p);
uchar4 __attribute__((overloadable)) vload4(size_t offset, const __private uchar *p);
short4 __attribute__((overloadable)) vload4(size_t offset, const __private short *p);
ushort4 __attribute__((overloadable)) vload4(size_t offset, const __private ushort *p);
int4 __attribute__((overloadable)) vload4(size_t offset, const __private int *p);
uint4 __attribute__((overloadable)) vload4(size_t offset, const __private uint *p);
long4 __attribute__((overloadable)) vload4(size_t offset, const __private long *p);
ulong4 __attribute__((overloadable)) vload4(size_t offset, const __private ulong *p);
```

```
float4 __attribute__((overloadable)) vload4(size_t offset, const __private float *p);
char8 __attribute__((overloadable)) vload8(size_t offset, const __private char *p);
uchar8 __attribute__((overloadable)) vload8(size_t offset, const __private uchar *p);
short8 __attribute__((overloadable)) vload8(size_t offset, const __private short *p);
ushort8 __attribute__((overloadable)) vload8(size_t offset, const __private ushort *p);
int8 __attribute__((overloadable)) vload8(size_t offset, const __private int *p);
uint8 __attribute__((overloadable)) vload8(size_t offset, const __private uint *p);
long8 __attribute__((overloadable)) vload8(size_t offset, const __private long *p);
ulong8 __attribute__((overloadable)) vload8(size_t offset, const __private ulong *p);
float8 __attribute__((overloadable)) vload8(size_t offset, const __private float *p);
char16 __attribute__((overloadable)) vload16(size_t offset, const __private char *p);
uchar16 __attribute__((overloadable)) vload16(size_t offset, const __private uchar *p);
short16 __attribute__((overloadable)) vload16(size_t offset, const __private short *p);
ushort16 __attribute__((overloadable)) vload16(size_t offset, const __private ushort *p);
int16 __attribute__((overloadable)) vload16(size_t offset, const __private int *p);
uint16 __attribute__((overloadable)) vload16(size_t offset, const __private uint *p);
long16 __attribute__((overloadable)) vload16(size_t offset, const __private long *p);
ulong16 __attribute__((overloadable)) vload16(size_t offset, const __private ulong *p);
float16 __attribute__((overloadable)) vload16(size_t offset, const __private float *p);
double2 __attribute__((overloadable)) vload2(size_t offset, const __global double *p);
double3 __attribute__((overloadable)) vload3(size_t offset, const __global double *p);
double4 __attribute__((overloadable)) vload4(size_t offset, const __global double *p);
double8 __attribute__((overloadable)) vload8(size_t offset, const __global double *p);
double16 __attribute__((overloadable)) vload16(size_t offset, const __global double *p);
double2 __attribute__((overloadable)) vload2(size_t offset, const __local double *p);
double3 __attribute__((overloadable)) vload3(size_t offset, const __local double *p);
double4 __attribute__((overloadable)) vload4(size_t offset, const __local double *p);
double8 __attribute__((overloadable)) vload8(size_t offset, const __local double *p);
double16 __attribute__((overloadable)) vload16(size_t offset, const __local double *p);
double2 __attribute__((overloadable)) vload2(size_t offset, const __private double *p);
double3 __attribute__((overloadable)) vload3(size_t offset, const __private double *p);
double4 __attribute__((overloadable)) vload4(size_t offset, const __private double *p);
double8 __attribute__((overloadable)) vload8(size_t offset, const __private double *p);
double16 __attribute__((overloadable)) vload16(size_t offset, const __private double *p);
//half __attribute__((overloadable)) vload(size_t offset, const __global half *p);
//half2 __attribute__((overloadable)) vload2(size_t offset, const __global half *p);
//half3 __attribute__((overloadable)) vload3(size_t offset, const __global half *p);
//half4 __attribute__((overloadable)) vload4(size_t offset, const __global half *p);
//half8 __attribute__((overloadable)) vload8(size_t offset, const __global half *p);
//half16 __attribute__((overloadable)) vload16(size_t offset, const __global half *p);
//half __attribute__((overloadable)) vload(size_t offset, const __local half *p);
//half2 __attribute__((overloadable)) vload2(size_t offset, const __local half *p);
//half3 __attribute__((overloadable)) vload3(size_t offset, const __local half *p);
//half4 __attribute__((overloadable)) vload4(size_t offset, const __local half *p);
//half8 __attribute__((overloadable)) vload8(size_t offset, const __local half *p);
//half16 __attribute__((overloadable)) vload16(size_t offset, const __local half *p);
//half __attribute__((overloadable)) vload(size_t offset, const __private half *p);
//half2 __attribute__((overloadable)) vload2(size_t offset, const __private half *p);
//half3 __attribute__((overloadable)) vload3(size_t offset, const __private half *p);
//half4 __attribute__((overloadable)) vload4(size_t offset, const __private half *p);
//half8 __attribute__((overloadable)) vload8(size_t offset, const __private half *p);
//half16 __attribute__((overloadable)) vload16(size_t offset, const __private half *p);
```

```
char2 __attribute__((overloadable)) vload2(size_t offset, const __constant char *p);
uchar2 __attribute__((overloadable)) vload2(size_t offset, const __constant uchar *p);
short2 __attribute__((overloadable)) vload2(size_t offset, const __constant short *p);
ushort2 __attribute__((overloadable)) vload2(size_t offset, const __constant ushort *p);
int2 __attribute__((overloadable)) vload2(size_t offset, const __constant int *p);
uint2 __attribute__((overloadable)) vload2(size_t offset, const __constant uint *p);
long2 __attribute__((overloadable)) vload2(size_t offset, const __constant long *p);
ulong2 __attribute__((overloadable)) vload2(size_t offset, const __constant ulong *p);
float2 __attribute__((overloadable)) vload2(size_t offset, const __constant float *p);
char3 __attribute__((overloadable)) vload3(size_t offset, const __constant char *p);
uchar3 __attribute__((overloadable)) vload3(size_t offset, const __constant uchar *p);
short3 __attribute__((overloadable)) vload3(size_t offset, const __constant short *p);
ushort3 __attribute__((overloadable)) vload3(size_t offset, const __constant ushort *p);
int3 __attribute__((overloadable)) vload3(size_t offset, const __constant int *p);
uint3 __attribute__((overloadable)) vload3(size_t offset, const __constant uint *p);
long3 __attribute__((overloadable)) vload3(size_t offset, const __constant long *p);
ulong3 __attribute__((overloadable)) vload3(size_t offset, const __constant ulong *p);
float3 __attribute__((overloadable)) vload3(size_t offset, const __constant float *p);
char4 __attribute__((overloadable)) vload4(size_t offset, const __constant char *p);
uchar4 __attribute__((overloadable)) vload4(size_t offset, const __constant uchar *p);
short4 __attribute__((overloadable)) vload4(size_t offset, const __constant short *p);
ushort4 __attribute__((overloadable)) vload4(size_t offset, const __constant ushort *p);
int4 __attribute__((overloadable)) vload4(size_t offset, const __constant int *p);
uint4 __attribute__((overloadable)) vload4(size_t offset, const __constant uint *p);
long4 __attribute__((overloadable)) vload4(size_t offset, const __constant long *p);
ulong4 __attribute__((overloadable)) vload4(size_t offset, const __constant ulong *p);
float4 __attribute__((overloadable)) vload4(size_t offset, const __constant float *p);
char8 __attribute__((overloadable)) vload8(size_t offset, const __constant char *p);
uchar8 __attribute__((overloadable)) vload8(size_t offset, const __constant uchar *p);
short8 __attribute__((overloadable)) vload8(size_t offset, const __constant short *p);
ushort8 __attribute__((overloadable)) vload8(size_t offset, const __constant ushort *p);
int8 __attribute__((overloadable)) vload8(size_t offset, const __constant int *p);
uint8 __attribute__((overloadable)) vload8(size_t offset, const __constant uint *p);
long8 __attribute__((overloadable)) vload8(size_t offset, const __constant long *p);
ulong8 __attribute__((overloadable)) vload8(size_t offset, const __constant ulong *p);
float8 __attribute__((overloadable)) vload8(size_t offset, const __constant float *p);
char16 __attribute__((overloadable)) vload16(size_t offset, const __constant char *p);
uchar16 __attribute__((overloadable)) vload16(size_t offset, const __constant uchar *p);
short16 __attribute__((overloadable)) vload16(size_t offset, const __constant short *p);
ushort16 __attribute__((overloadable)) vload16(size_t offset, const __constant ushort *p);
int16 __attribute__((overloadable)) vload16(size_t offset, const __constant int *p);
uint16 __attribute__((overloadable)) vload16(size_t offset, const __constant uint *p);
long16 __attribute__((overloadable)) vload16(size_t offset, const __constant long *p);
ulong16 __attribute__((overloadable)) vload16(size_t offset, const __constant
```

```
ulong *p);
float16 __attribute__((overloadable)) vload16(size_t offset, const __constant float *p);
double2 __attribute__((overloadable)) vload2(size_t offset, const __constant double *p);
double3 __attribute__((overloadable)) vload3(size_t offset, const __constant double *p);
double4 __attribute__((overloadable)) vload4(size_t offset, const __constant double *p);
double8 __attribute__((overloadable)) vload8(size_t offset, const __constant double *p);
double16 __attribute__((overloadable)) vload16(size_t offset, const __constant double *p);
//half __attribute__((overloadable)) vload(size_t offset, const __constant half *p);
//half2 __attribute__((overloadable)) vload2(size_t offset, const __constant half *p);
//half3 __attribute__((overloadable)) vload3(size_t offset, const __constant half *p);
//half4 __attribute__((overloadable)) vload4(size_t offset, const __constant half *p);
//half8 __attribute__((overloadable)) vload8(size_t offset, const __constant half *p);
//half16 __attribute__((overloadable)) vload16(size_t offset, const __constant half *p);

/**
 * Write sizeof (gentypen) bytes given by data
 * to address (p + (offset * n)). The address
 * computed as (p + (offset * n)) must be 8-bit
 * aligned if gentype is char, uchar; 16-bit
 * aligned if gentype is short, ushort; 32-bit
 * aligned if gentype is int, uint, float; 64-bit
 * aligned if gentype is long, ulong.
 */
void __attribute__((overloadable)) vstore2(char2 data, size_t offset, __global char *p);
void __attribute__((overloadable)) vstore2(uchar2 data, size_t offset, __global uchar *p);
void __attribute__((overloadable)) vstore2(short2 data, size_t offset, __global short *p);
void __attribute__((overloadable)) vstore2(ushort2 data, size_t offset, __global ushort *p);
void __attribute__((overloadable)) vstore2(int2 data, size_t offset, __global int *p);
void __attribute__((overloadable)) vstore2(uint2 data, size_t offset, __global uint *p);
void __attribute__((overloadable)) vstore2(long2 data, size_t offset, __global long *p);
void __attribute__((overloadable)) vstore2(ulong2 data, size_t offset, __global ulong *p);
void __attribute__((overloadable)) vstore2(float2 data, size_t offset, __global float *p);
void __attribute__((overloadable)) vstore3(char3 data, size_t offset, __global char *p);
void __attribute__((overloadable)) vstore3(uchar3 data, size_t offset, __global uchar *p);
void __attribute__((overloadable)) vstore3(short3 data, size_t offset, __global short *p);
void __attribute__((overloadable)) vstore3(ushort3 data, size_t offset, __global ushort *p);
void __attribute__((overloadable)) vstore3(int3 data, size_t offset, __global int *p);
void __attribute__((overloadable)) vstore3(uint3 data, size_t offset, __global uint *p);
void __attribute__((overloadable)) vstore3(long3 data, size_t offset, __global long *p);
void __attribute__((overloadable)) vstore3(ulong3 data, size_t offset, __global ulong *p);
void __attribute__((overloadable)) vstore3(float3 data, size_t offset, __global float *p);
void __attribute__((overloadable)) vstore4(char4 data, size_t offset, __global char *p);
void __attribute__((overloadable)) vstore4(uchar4 data, size_t offset, __global uchar *p);
void __attribute__((overloadable)) vstore4(short4 data, size_t offset, __global short *p);
void __attribute__((overloadable)) vstore4(ushort4 data, size_t offset, __global ushort *p);
void __attribute__((overloadable)) vstore4(int4 data, size_t offset, __global int *p);
void __attribute__((overloadable)) vstore4(uint4 data, size_t offset, __global uint *p);
void __attribute__((overloadable)) vstore4(long4 data, size_t offset, __global long *p);
void __attribute__((overloadable)) vstore4(ulong4 data, size_t offset, __global ulong *p);
void __attribute__((overloadable)) vstore4(float4 data, size_t offset, __global float *p);
void __attribute__((overloadable)) vstore8(char8 data, size_t offset, __global char *p);
void __attribute__((overloadable)) vstore8(uchar8 data, size_t offset, __global uchar *p);
```

```
void __attribute__((overloadable)) vstore8(short8 data, size_t offset, __global short *p);
void __attribute__((overloadable)) vstore8(ushort8 data, size_t offset, __global ushort *p);
void __attribute__((overloadable)) vstore8(int8 data, size_t offset, __global int *p);
void __attribute__((overloadable)) vstore8(uint8 data, size_t offset, __global uint *p);
void __attribute__((overloadable)) vstore8(long8 data, size_t offset, __global long *p);
void __attribute__((overloadable)) vstore8(ulong8 data, size_t offset, __global ulong *p);
void __attribute__((overloadable)) vstore8(float8 data, size_t offset, __global float *p);
void __attribute__((overloadable)) vstore16(char16 data, size_t offset, __global char *p);
void __attribute__((overloadable)) vstore16(uchar16 data, size_t offset, __global uchar *p);
void __attribute__((overloadable)) vstore16(short16 data, size_t offset, __global short *p);
void __attribute__((overloadable)) vstore16(ushort16 data, size_t offset, __global ushort *p);
void __attribute__((overloadable)) vstore16(int16 data, size_t offset, __global int *p);
void __attribute__((overloadable)) vstore16(uint16 data, size_t offset, __global uint *p);
void __attribute__((overloadable)) vstore16(long16 data, size_t offset, __global long *p);
void __attribute__((overloadable)) vstore16(ulong16 data, size_t offset, __global ulong *p);
void __attribute__((overloadable)) vstore16(float16 data, size_t offset, __global float *p);
void __attribute__((overloadable)) vstore2(char2 data, size_t offset, __local char *p);
void __attribute__((overloadable)) vstore2(uchar2 data, size_t offset, __local uchar *p);
void __attribute__((overloadable)) vstore2(short2 data, size_t offset, __local short *p);
void __attribute__((overloadable)) vstore2(ushort2 data, size_t offset, __local ushort *p);
void __attribute__((overloadable)) vstore2(int2 data, size_t offset, __local int *p);
void __attribute__((overloadable)) vstore2(uint2 data, size_t offset, __local uint *p);
void __attribute__((overloadable)) vstore2(long2 data, size_t offset, __local long *p);
void __attribute__((overloadable)) vstore2(ulong2 data, size_t offset, __local ulong *p);
void __attribute__((overloadable)) vstore2(float2 data, size_t offset, __local float *p);
void __attribute__((overloadable)) vstore3(char3 data, size_t offset, __local char *p);
void __attribute__((overloadable)) vstore3(uchar3 data, size_t offset, __local uchar *p);
void __attribute__((overloadable)) vstore3(short3 data, size_t offset, __local short *p);
void __attribute__((overloadable)) vstore3(ushort3 data, size_t offset, __local ushort *p);
void __attribute__((overloadable)) vstore3(int3 data, size_t offset, __local int *p);
void __attribute__((overloadable)) vstore3(uint3 data, size_t offset, __local uint *p);
void __attribute__((overloadable)) vstore3(long3 data, size_t offset, __local long *p);
void __attribute__((overloadable)) vstore3(ulong3 data, size_t offset, __local ulong *p);
void __attribute__((overloadable)) vstore3(float3 data, size_t offset, __local float *p);
void __attribute__((overloadable)) vstore4(char4 data, size_t offset, __local char *p);
void __attribute__((overloadable)) vstore4(uchar4 data, size_t offset, __local uchar *p);
void __attribute__((overloadable)) vstore4(short4 data, size_t offset, __local short *p);
void __attribute__((overloadable)) vstore4(ushort4 data, size_t offset, __local ushort *p);
void __attribute__((overloadable)) vstore4(int4 data, size_t offset, __local int *p);
void __attribute__((overloadable)) vstore4(uint4 data, size_t offset, __local uint *p);
void __attribute__((overloadable)) vstore4(long4 data, size_t offset, __local long *p);
void __attribute__((overloadable)) vstore4(ulong4 data, size_t offset, __local ulong *p);
void __attribute__((overloadable)) vstore4(float4 data, size_t offset, __local float *p);
void __attribute__((overloadable)) vstore8(char8 data, size_t
```

```
offset, __local char *p);
void __attribute__((overloadable)) vstore8(uchar8 data, size_t offset, __local uchar *p);
void __attribute__((overloadable)) vstore8(short8 data, size_t offset, __local short *p);
void __attribute__((overloadable)) vstore8(ushort8 data, size_t offset, __local ushort *p);
void __attribute__((overloadable)) vstore8(int8 data, size_t offset, __local int *p);
void __attribute__((overloadable)) vstore8(uint8 data, size_t offset, __local uint *p);
void __attribute__((overloadable)) vstore8(long8 data, size_t offset, __local long *p);
void __attribute__((overloadable)) vstore8(ulong8 data, size_t offset, __local ulong *p);
void __attribute__((overloadable)) vstore8(float8 data, size_t offset, __local float *p);
void __attribute__((overloadable)) vstore16(char16 data, size_t offset, __local char *p);
void __attribute__((overloadable)) vstore16(uchar16 data, size_t offset, __local uchar *p);
void __attribute__((overloadable)) vstore16(short16 data, size_t offset, __local short *p);
void __attribute__((overloadable)) vstore16(ushort16 data, size_t offset, __local ushort *p);
void __attribute__((overloadable)) vstore16(int16 data, size_t offset, __local int *p);
void __attribute__((overloadable)) vstore16(uint16 data, size_t offset, __local uint *p);
void __attribute__((overloadable)) vstore16(long16 data, size_t offset, __local long *p);
void __attribute__((overloadable)) vstore16(ulong16 data, size_t offset, __local ulong *p);
void __attribute__((overloadable)) vstore16(float16 data, size_t offset, __local float *p);
void __attribute__((overloadable)) vstore2(char2 data, size_t offset, __private char *p);
void __attribute__((overloadable)) vstore2(uchar2 data, size_t offset, __private uchar *p);
void __attribute__((overloadable)) vstore2(short2 data, size_t offset, __private short *p);
void __attribute__((overloadable)) vstore2(ushort2 data, size_t offset, __private ushort *p);
void __attribute__((overloadable)) vstore2(int2 data, size_t offset, __private int *p);
void __attribute__((overloadable)) vstore2(uint2 data, size_t offset, __private uint *p);
void __attribute__((overloadable)) vstore2(long2 data, size_t offset, __private long *p);
void __attribute__((overloadable)) vstore2(ulong2 data, size_t offset, __private ulong *p);
void __attribute__((overloadable)) vstore2(float2 data, size_t offset, __private float *p);
void __attribute__((overloadable)) vstore3(char3 data, size_t offset, __private char *p);
void __attribute__((overloadable)) vstore3(uchar3 data, size_t offset, __private uchar *p);
void __attribute__((overloadable)) vstore3(short3 data, size_t offset, __private short *p);
void __attribute__((overloadable)) vstore3(ushort3 data, size_t offset, __private ushort *p);
void __attribute__((overloadable)) vstore3(int3 data, size_t offset, __private int *p);
void __attribute__((overloadable)) vstore3(uint3 data, size_t offset, __private uint *p);
void __attribute__((overloadable)) vstore3(long3 data, size_t offset, __private long *p);
void __attribute__((overloadable)) vstore3(ulong3 data, size_t offset, __private ulong *p);
void __attribute__((overloadable)) vstore3(float3 data, size_t offset, __private float *p);
void __attribute__((overloadable)) vstore4(char4 data, size_t offset, __private char *p);
void __attribute__((overloadable)) vstore4(uchar4 data, size_t offset, __private uchar *p);
void __attribute__((overloadable)) vstore4(short4 data, size_t offset, __private short *p);
void __attribute__((overloadable)) vstore4(ushort4 data, size_t offset, __private ushort *p);
void __attribute__((overloadable)) vstore4(int4 data, size_t offset, __private int *p);
void __attribute__((overloadable)) vstore4(uint4 data, size_t offset, __private uint *p);
void __attribute__((overloadable)) vstore4(long4 data, size_t offset, __private long *p);
void __attribute__((overloadable)) vstore4(ulong4 data, size_t offset, __private ulong *p);
```

```
void __attribute__((overloadable)) vstore4(float4 data, size_t offset, __private float *p);
void __attribute__((overloadable)) vstore8(char8 data, size_t offset, __private char *p);
void __attribute__((overloadable)) vstore8(uchar8 data, size_t offset, __private uchar *p);
void __attribute__((overloadable)) vstore8(short8 data, size_t offset, __private short *p);
void __attribute__((overloadable)) vstore8(ushort8 data, size_t offset, __private ushort *p);
void __attribute__((overloadable)) vstore8(int8 data, size_t offset, __private int *p);
void __attribute__((overloadable)) vstore8(uint8 data, size_t offset, __private uint *p);
void __attribute__((overloadable)) vstore8(long8 data, size_t offset, __private long *p);
void __attribute__((overloadable)) vstore8(ulong8 data, size_t offset, __private ulong *p);
void __attribute__((overloadable)) vstore8(float8 data, size_t offset, __private float *p);
void __attribute__((overloadable)) vstore16(char16 data, size_t offset, __private char *p);
void __attribute__((overloadable)) vstore16(uchar16 data, size_t offset, __private uchar *p);
void __attribute__((overloadable)) vstore16(short16 data, size_t offset, __private short *p);
void __attribute__((overloadable)) vstore16(ushort16 data, size_t offset, __private ushort *p);
void __attribute__((overloadable)) vstore16(int16 data, size_t offset, __private int *p);
void __attribute__((overloadable)) vstore16(uint16 data, size_t offset, __private uint *p);
void __attribute__((overloadable)) vstore16(long16 data, size_t offset, __private long *p);
void __attribute__((overloadable)) vstore16(ulong16 data, size_t offset, __private ulong *p);
void __attribute__((overloadable)) vstore16(float16 data, size_t offset, __private float *p);
void __attribute__((overloadable)) vstore2(double2 data, size_t offset, __global double *p);
void __attribute__((overloadable)) vstore3(double3 data, size_t offset, __global double *p);
void __attribute__((overloadable)) vstore4(double4 data, size_t offset, __global double *p);
void __attribute__((overloadable)) vstore8(double8 data, size_t offset, __global double *p);
void __attribute__((overloadable)) vstore16(double16 data, size_t offset, __global double *p);
void __attribute__((overloadable)) vstore2(double2 data, size_t offset, __local double *p);
void __attribute__((overloadable)) vstore3(double3 data, size_t offset, __local double *p);
void __attribute__((overloadable)) vstore4(double4 data, size_t offset, __local double *p);
void __attribute__((overloadable)) vstore8(double8 data, size_t offset, __local double *p);
void __attribute__((overloadable)) vstore16(double16 data, size_t offset, __local double *p);
void __attribute__((overloadable)) vstore2(double2 data, size_t offset, __private double *p);
void __attribute__((overloadable)) vstore3(double3 data, size_t offset, __private double *p);
void __attribute__((overloadable)) vstore4(double4 data, size_t offset, __private double *p);
void __attribute__((overloadable)) vstore8(double8 data, size_t offset, __private double *p);
void __attribute__((overloadable)) vstore16(double16 data, size_t offset, __private double *p);
//void __attribute__((overloadable)) vstore(half data, size_t offset, __global half *p);
//void __attribute__((overloadable)) vstore2(half2 data, size_t offset, __global half *p);
//void __attribute__((overloadable)) vstore3(half3 data, size_t offset, __global half *p);
//void __attribute__((overloadable)) vstore4(half4 data, size_t offset, __global half *p);
//void __attribute__((overloadable)) vstore8(half8 data, size_t offset, __global half *p);
//void __attribute__((overloadable)) vstore16(half16 data, size_t offset, __global half *p);
//void __attribute__((overloadable)) vstore(half data, size_t offset, __local half *p);
//void __attribute__((overloadable)) vstore2(half2 data, size_t offset, __local half *p);
//void __attribute__((overloadable)) vstore3(half3 data, size_t offset, __local half *p);
//void __attribute__((overloadable)) vstore4(half4 data, size_t offset, __local half *p);
//void __attribute__((overloadable)) vstore8(half8 data, size_t offset, __local half *p);
//void __attribute__((overloadable)) vstore16(half16 data, size_t offset, __local half *p);
//void __attribute__((overloadable)) vstore(half data, size_t offset, __private half *p);
//void __attribute__((overloadable)) vstore2(half2 data, size_t offset, __private half *p);
//void __attribute__((overloadable)) vstore3(half3 data, size_t offset, __private half *p);
//void __attribute__((overloadable)) vstore4(half4 data, size_t offset, __private half *p);
//void __attribute__((overloadable)) vstore8(half8 data, size_t offset, __private half *p);
//void __attribute__((overloadable)) vstore16(half16 data, size_t offset, __private half *p);
```

```
/**
 * Read sizeof (half) bytes of data from address
 * (p + offset). The data read is interpreted as a
 * half value. The half value is converted to a
 * float value and the float value is returned.
 * The read address computed as (p + offset)
 * must be 16-bit aligned.
 */
float __attribute__((overloadable)) vload_half(size_t offset, const __global half *p);
float __attribute__((overloadable)) vload_half(size_t offset, const __local half *p);
float __attribute__((overloadable)) vload_half(size_t offset, const __private half *p);
float __attribute__((overloadable)) vload_half(size_t offset, const __constant half *p);

/**
 * Read sizeof (halfn) bytes of data from address
 * (p + (offset * n)). The data read is interpreted
 * as a halfn value. The halfn value read is
 * converted to a floatn value and the floatn
 * value is returned. The read address computed
 * as (p + (offset * n)) must be 16-bit aligned.
 */
float2 __attribute__((overloadable)) vload_half2(size_t offset, const __global half *p);
float3 __attribute__((overloadable)) vload_half3(size_t offset, const __global half *p);
float4 __attribute__((overloadable)) vload_half4(size_t offset, const __global half *p);
float8 __attribute__((overloadable)) vload_half8(size_t offset, const __global half *p);
float16 __attribute__((overloadable)) vload_half16(size_t offset, const __global half *p);
float2 __attribute__((overloadable)) vload_half2(size_t offset, const __local half *p);
float3 __attribute__((overloadable)) vload_half3(size_t offset, const __local half *p);
float4 __attribute__((overloadable)) vload_half4(size_t offset, const __local half *p);
float8 __attribute__((overloadable)) vload_half8(size_t offset, const __local half *p);
float16 __attribute__((overloadable)) vload_half16(size_t offset, const __local half *p);
float2 __attribute__((overloadable)) vload_half2(size_t offset, const __private half *p);
float3 __attribute__((overloadable)) vload_half3(size_t offset, const __private half *p);
float4 __attribute__((overloadable)) vload_half4(size_t offset, const __private half *p);
float8 __attribute__((overloadable)) vload_half8(size_t offset, const __private half *p);
float16 __attribute__((overloadable)) vload_half16(size_t offset, const __private half *p);
float2 __attribute__((overloadable)) vload_half2(size_t offset, const __constant half *p);
float3 __attribute__((overloadable)) vload_half3(size_t offset, const __constant half *p);
float4 __attribute__((overloadable)) vload_half4(size_t offset, const __constant half *p);
float8 __attribute__((overloadable)) vload_half8(size_t offset, const __constant half *p);
float16 __attribute__((overloadable)) vload_half16(size_t offset, const __constant half *p);

/**
 * The float value given by data is first
 * converted to a half value using the appropriate
 * rounding mode. The half value is then written
 * to address computed as (p + offset). The
 * address computed as (p + offset) must be 16-
 * bit aligned.
 * vstore_half use the current rounding mode.
 * The default current rounding mode is round to
 * nearest even.
 */
void __attribute__((overloadable)) vstore_half(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rte(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rtz(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rtp(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rtn(float data, size_t
```

```
offset, __global half *p);
void __attribute__((overloadable)) vstore_half(float data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rte(float data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rtz(float data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rtp(float data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rtn(float data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half(float data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rte(float data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rtz(float data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rtp(float data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rtn(float data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half(double data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rte(double data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rtz(double data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rtp(double data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half_rtn(double data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half(double data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rte(double data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rtz(double data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rtp(double data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half_rtn(double data, size_t
offset, __local half *p);
void __attribute__((overloadable)) vstore_half(double data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rte(double data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rtz(double data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rtp(double data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half_rtn(double data, size_t
offset, __private half *p);

/**
 * The floatn value given by data is converted to
 * a halfn value using the appropriate rounding
 * mode. The halfn value is then written to
 * address computed as (p + (offset * n)). The
 * address computed as (p + (offset * n)) must be
 * 16-bit aligned.
 * vstore_halfn uses the current rounding mode.
 * The default current rounding mode is round to
 * nearest even.
 */
void __attribute__((overloadable)) vstore_half2(float2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3(float3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4(float4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8(float8 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half16(float16 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rte(float2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rte(float3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rte(float4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rte(float8 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rte(float16 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rtz(float2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rtz(float3 data, size_t
offset, __global half *p);
```

```
void __attribute__((overloadable)) vstore_half4_rtz(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rtz(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rtz(float16 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rtp(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rtp(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rtp(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rtp(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rtp(float16 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rtn(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rtn(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rtn(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rtn(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rtn(float16 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half2(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16(float16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rte(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rte(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rte(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rte(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rte(float16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rtz(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rtz(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rtz(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rtz(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rtz(float16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rtp(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rtp(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rtp(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rtp(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rtp(float16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rtn(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rtn(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rtn(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rtn(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rtn(float16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2(float2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half3(float3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half4(float4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half8(float8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half16(float16 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rte(float2 data, size_t
```

```
offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rte(float3 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rte(float4 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rte(float8 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rte(float16 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rtz(float2 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rtz(float3 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rtz(float4 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rtz(float8 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rtz(float16 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rtp(float2 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rtp(float3 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rtp(float4 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rtp(float8 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rtp(float16 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rtn(float2 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rtn(float3 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rtn(float4 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rtn(float8 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rtn(float16 data, size_t
offset, __private half *p);
void __attribute__((overloadable)) vstore_half2(double2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3(double3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4(double4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8(double8 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half16(double16 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rte(double2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rte(double3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rte(double4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rte(double8 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rte(double16 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rtz(double2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rtz(double3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rtz(double4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rtz(double8 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rtz(double16 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rtp(double2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rtp(double3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rtp(double4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rtp(double8 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half16_rtp(double16 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half2_rtn(double2 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half3_rtn(double3 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half4_rtn(double4 data, size_t
offset, __global half *p);
void __attribute__((overloadable)) vstore_half8_rtn(double8 data, size_t
offset, __global half *p);
```

```
void __attribute__((overloadable)) vstore_half16_rtn(double16 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstore_half2(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16(double16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rte(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rte(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rte(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rte(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rte(double16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rtz(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rtz(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rtz(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rtz(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rtz(double16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rtp(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rtp(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rtp(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rtp(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rtp(double16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2_rtn(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half3_rtn(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half4_rtn(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half8_rtn(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half16_rtn(double16 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstore_half2(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half3(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half4(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half8(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half16(double16 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rte(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rte(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rte(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rte(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rte(double16 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rtz(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rtz(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rtz(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rtz(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rtz(double16 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rtp(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rtp(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rtp(double4 data, size_t
```

```
offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rtp(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rtp(double16 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half2_rtn(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half3_rtn(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half4_rtn(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half8_rtn(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstore_half16_rtn(double16 data, size_t offset, __private half *p);

/**
 * For n = 1, 2, 4, 8 and 16 read sizeof (halfn)
 * bytes of data from address (p + (offset * n)).
 * The data read is interpreted as a halfn value.
 * The halfn value read is converted to a floatn
 * value and the floatn value is returned.
 * The address computed as (p + (offset * n))
 * must be aligned to sizeof (halfn) bytes.
 * For n = 3, vloada_half3 reads a half3 from
 * address (p + (offset * 4)) and returns a float3.
 * The address computed as (p + (offset * 4))
 * must be aligned to sizeof (half) * 4 bytes.
 */
float __attribute__((overloadable)) vloada_half(size_t offset, const __global half *p);
float2 __attribute__((overloadable)) vloada_half2(size_t offset, const __global half *p);
float3 __attribute__((overloadable)) vloada_half3(size_t offset, const __global half *p);
float4 __attribute__((overloadable)) vloada_half4(size_t offset, const __global half *p);
float8 __attribute__((overloadable)) vloada_half8(size_t offset, const __global half *p);
float16 __attribute__((overloadable)) vloada_half16(size_t offset, const __global half *p);
float __attribute__((overloadable)) vloada_half(size_t offset, const __local half *p);
float2 __attribute__((overloadable)) vloada_half2(size_t offset, const __local half *p);
float3 __attribute__((overloadable)) vloada_half3(size_t offset, const __local half *p);
float4 __attribute__((overloadable)) vloada_half4(size_t offset, const __local half *p);
float8 __attribute__((overloadable)) vloada_half8(size_t offset, const __local half *p);
float16 __attribute__((overloadable)) vloada_half16(size_t offset, const __local half *p);
float __attribute__((overloadable)) vloada_half(size_t offset, const __private half *p);
float2 __attribute__((overloadable)) vloada_half2(size_t offset, const __private half *p);
float3 __attribute__((overloadable)) vloada_half3(size_t offset, const __private half *p);
float4 __attribute__((overloadable)) vloada_half4(size_t offset, const __private half *p);
float8 __attribute__((overloadable)) vloada_half8(size_t offset, const __private half *p);
float16 __attribute__((overloadable)) vloada_half16(size_t offset, const __private half *p);
float __attribute__((overloadable)) vloada_half(size_t offset, const __constant half *p);
float2 __attribute__((overloadable)) vloada_half2(size_t offset, const __constant half *p);
float3 __attribute__((overloadable)) vloada_half3(size_t offset, const __constant half *p);
float4 __attribute__((overloadable)) vloada_half4(size_t offset, const __constant half *p);
float8 __attribute__((overloadable)) vloada_half8(size_t offset, const __constant half *p);
float16 __attribute__((overloadable)) vloada_half16(size_t offset, const __constant half *p);

/**
 * The floatn value given by data is converted to
 * a halfn value using the appropriate rounding
 * mode.
 * For n = 1, 2, 4, 8 and 16, the halfn value is
 * written to the address computed as (p + (offset
 * * n)). The address computed as (p + (offset *
 * n)) must be aligned to sizeof (halfn) bytes.
 * For n = 3, the half3 value is written to the
```

```
 * address computed as (p + (offset * 4)). The
 * address computed as (p + (offset * 4)) must be
 * aligned to sizeof (half) * 4 bytes.
 * vstorea_halfn uses the current rounding
 * mode. The default current rounding mode is
 * round to nearest even.
 */
void __attribute__((overloadable)) vstorea_half(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16(float16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rte(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rte(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rte(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rte(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rte(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rte(float16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rtz(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rtz(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rtz(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rtz(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rtz(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rtz(float16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rtp(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rtp(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rtp(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rtp(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rtp(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rtp(float16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rtn(float data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rtn(float2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rtn(float3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rtn(float4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rtn(float8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rtn(float16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half(float data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16(float16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rte(float data, size_t offset, __local half *p);
```

```
void __attribute__((overloadable)) vstorea_half2_rte(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rte(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rte(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rte(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rte(float16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rtz(float data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rtz(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rtz(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rtz(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rtz(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rtz(float16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rtp(float data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rtp(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rtp(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rtp(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rtp(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rtp(float16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rtn(float data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rtn(float2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rtn(float3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rtn(float4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rtn(float8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rtn(float16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half(float data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2(float2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3(float3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4(float4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8(float8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16(float16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rte(float data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rte(float2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rte(float3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rte(float4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rte(float8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rte(float16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rtz(float data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rtz(float2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rtz(float3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rtz(float4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rtz(float8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rtz(float16 data, size_t
```

```
offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rtp(float data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rtp(float2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rtp(float3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rtp(float4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rtp(float8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rtp(float16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rtn(float data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rtn(float2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rtn(float3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rtn(float4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rtn(float8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rtn(float16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half(double data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2(double2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3(double3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4(double4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8(double8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16(double16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rte(double data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rte(double2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rte(double3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rte(double4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rte(double8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rte(double16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rtz(double data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rtz(double2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rtz(double3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rtz(double4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rtz(double8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rtz(double16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rtp(double data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rtp(double2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rtp(double3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rtp(double4 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rtp(double8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rtp(double16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half_rtn(double data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half2_rtn(double2 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half3_rtn(double3 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half4_rtn(double4 data, size_t
```

```
offset, __global half *p);
void __attribute__((overloadable)) vstorea_half8_rtn(double8 data, size_t offset, __global half *p);
void __attribute__((overloadable)) vstorea_half16_rtn(double16 data, size_t offset, __global half *p);

void __attribute__((overloadable)) vstorea_half(double data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16(double16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rte(double data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rte(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rte(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rte(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rte(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rte(double16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rtz(double data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rtz(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rtz(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rtz(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rtz(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rtz(double16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rtp(double data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rtp(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rtp(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rtp(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rtp(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rtp(double16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half_rtn(double data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half2_rtn(double2 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half3_rtn(double3 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half4_rtn(double4 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half8_rtn(double8 data, size_t offset, __local half *p);
void __attribute__((overloadable)) vstorea_half16_rtn(double16 data, size_t offset, __local half *p);

void __attribute__((overloadable)) vstorea_half(double data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16(double16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rte(double data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rte(double2 data, size_t
```

```
offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rte(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rte(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rte(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rte(double16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rtz(double data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rtz(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rtz(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rtz(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rtz(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rtz(double16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rtp(double data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rtp(double2 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rtp(double3 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rtp(double4 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rtp(double8 data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rtp(double16 data, size_t offset, __private half *p);

void __attribute__((overloadable)) vstorea_half_rtn(double data, size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half2_rtn(double2 data,size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half3_rtn(double3 data,size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half4_rtn(double4 data,size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half8_rtn(double8 data,size_t offset, __private half *p);
void __attribute__((overloadable)) vstorea_half16_rtn(double16 data,size_t offset, __private half *p);

// Synchronization functions

/**
 * All work-items in a work-group executing the kernel
 * on a processor must execute this function before any
 * are allowed to continue execution beyond the barrier.
 * This function must be encountered by all work-items in
 * a work-group executing the kernel.
 * If barrier is inside a conditional statement, then all
 * work-items must enter the conditional if any work-item
 * enters the conditional statement and executes the
 * barrier.
 * If barrer is inside a loop, all work-items must execute
 * the barrier for each iteration of the loop before any are
 * allowed to continue execution beyond the barrier.
 * The barrier function also queues a memory fence
 * (reads and writes) to ensure correct ordering of
 * memory operations to local or global memory.
 * The flags argument specifies the memory address space
 * and can be set to a combination of the following literal
 * values.
 * CLK_LOCAL_MEM_FENCE - The barrier function
 * will either flush any variables stored in local memory
 * or queue a memory fence to ensure correct ordering of
 * memory operations to local memory.
 * CLK_GLOBAL_MEM_FENCE - The barrier function
 * will queue a memory fence to ensure correct ordering
 * of memory operations to global memory. This can be
 * useful when work-items, for example, write to buffer or
 * image objects and then want to read the updated data.
 */

typedef uint cl_mem_fence_flags;

void __attribute__((overloadable)) barrier(cl_mem_fence_flags flags);

// Explicit memory fence functions

```

```
/**
 * Orders loads and stores of a work-item
 * executing a kernel. This means that loads
 * and stores preceding the mem_fence will
 * be committed to memory before any loads
 * and stores following the mem_fence.
 * The flags argument specifies the memory
 * address space and can be set to a
 * combination of the following literal
 * values:
 * CLK_LOCAL_MEM_FENCE
 * CLK_GLOBAL_MEM_FENCE.
 */
void __attribute__((overloadable)) mem_fence(cl_mem_fence_flags flags);

/**
 * Read memory barrier that orders only
 * loads.
 * The flags argument specifies the memory
 * address space and can be set to to a
 * combination of the following literal
 * values:
 * CLK_LOCAL_MEM_FENCE
 * CLK_GLOBAL_MEM_FENCE.
 */
void __attribute__((overloadable)) read_mem_fence(cl_mem_fence_flags flags);

/**
 * Write memory barrier that orders only
 * stores.
 * The flags argument specifies the memory
 * address space and can be set to to a
 * combination of the following literal
 * values:
 * CLK_LOCAL_MEM_FENCE
 * CLK_GLOBAL_MEM_FENCE.
 */
void __attribute__((overloadable)) write_mem_fence(cl_mem_fence_flags flags);

// Flag values for barrier, mem_fence, read_mem_fence, write_mem_fence

/**
 * Queue a memory fence to ensure correct
 * ordering of memory operations to local memory
 */
#define CLK_LOCAL_MEM_FENCE    0x1

/**
 * Queue a memory fence to ensure correct
 * ordering of memory operations to global memory
 */
#define CLK_GLOBAL_MEM_FENCE   0x2


// Async copies from global to local memory, local to global memory, and prefetch

/**
 * event_t async_work_group_copy (
 * __global gentype *dst,
 * const __local gentype *src,
 * size_t num_elements,
 * event_t event)
 * Perform an async copy of num_elements
 * gentype elements from src to dst. The async
 * copy is performed by all work-items in a workgroup
 * and this built-in function must therefore
 * be encountered by all work-items in a workgroup
 * executing the kernel with the same
 * argument values; otherwise the results are
 * undefined.
 * Returns an event object that can be used by
 * wait_group_events to wait for the async copy
 * to finish. The event argument can also be used
 * to associate the async_work_group_copy with
 * a previous async copy allowing an event to be
 * shared by multiple async copies; otherwise event
 * should be zero.
 * If event argument is non-zero, the event object
 * supplied in event argument will be returned.
 * This function does not perform any implicit
 * synchronization of source data such as using a
 * barrier before performing the copy.
 */
event_t __attribute__((overloadable)) async_work_group_copy(__local char *dst, const __global
    char *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local
    uchar *dst, const __global uchar *src, size_t num_elements, event_t
```

```
event);
event_t __attribute__((overloadable)) async_work_group_copy(__local short *dst, const __global short *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ushort *dst, const __global ushort *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local int *dst, const __global int *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uint *dst, const __global uint *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local long *dst, const __global long *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ulong *dst, const __global ulong *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local float *dst, const __global float *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local char2 *dst, const __global char2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uchar2 *dst, const __global uchar2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local short2 *dst, const __global short2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ushort2 *dst, const __global ushort2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local int2 *dst, const __global int2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uint2 *dst, const __global uint2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local long2 *dst, const __global long2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ulong2 *dst, const __global ulong2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local float2 *dst, const __global float2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local char3 *dst, const __global char3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uchar3 *dst, const __global uchar3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local short3 *dst, const __global short3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ushort3 *dst, const __global ushort3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local int3 *dst, const __global int3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uint3 *dst, const __global uint3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local long3 *dst, const __global long3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ulong3 *dst, const __global ulong3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local float3 *dst, const __global float3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local char4 *dst, const __global char4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uchar4 *dst, const __global uchar4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local short4 *dst, const __global short4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ushort4 *dst, const __global ushort4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local int4 *dst, const __global int4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uint4 *dst, const __global uint4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local long4 *dst, const __global long4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ulong4 *dst, const __global ulong4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local float4 *dst, const __global float4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local char8 *dst, const __global char8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uchar8 *dst, const __global uchar8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local short8 *dst, const __global short8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ushort8 *dst, const __global ushort8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local int8 *dst, const __global int8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uint8 *dst, const __global uint8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local long8 *dst, const __global long8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ulong8 *dst, const
```

```
__global ulong8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local float8 *dst, const __global float8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local char16 *dst, const __global char16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uchar16 *dst, const __global uchar16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local short16 *dst, const __global short16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ushort16 *dst, const __global ushort16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local int16 *dst, const __global int16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local uint16 *dst, const __global uint16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local long16 *dst, const __global long16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local ulong16 *dst, const __global ulong16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local float16 *dst, const __global float16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global char *dst, const __local char *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uchar *dst, const __local uchar *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global short *dst, const __local short *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ushort *dst, const __local ushort *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global int *dst, const __local int *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uint *dst, const __local uint *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global long *dst, const __local long *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ulong *dst, const __local ulong *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global float *dst, const __local float *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global char2 *dst, const __local char2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uchar2 *dst, const __local uchar2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global short2 *dst, const __local short2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ushort2 *dst, const __local ushort2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global int2 *dst, const __local int2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uint2 *dst, const __local uint2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global long2 *dst, const __local long2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ulong2 *dst, const __local ulong2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global float2 *dst, const __local float2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global char3 *dst, const __local char3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uchar3 *dst, const __local uchar3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global short3 *dst, const __local short3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ushort3 *dst, const __local ushort3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global int3 *dst, const __local int3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uint3 *dst, const __local uint3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global long3 *dst, const __local long3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ulong3 *dst, const __local ulong3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global float3 *dst, const __local float3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global char4 *dst, const __local char4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uchar4 *dst, const __local uchar4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global short4 *dst, const __local short4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ushort4 *dst, const __local ushort4 *src, size_t num_elements, event_t event);
```

```
event_t __attribute__((overloadable)) async_work_group_copy(__global int4 *dst, const __local int4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uint4 *dst, const __local uint4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global long4 *dst, const __local long4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ulong4 *dst, const __local ulong4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global float4 *dst, const __local float4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global char8 *dst, const __local char8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uchar8 *dst, const __local uchar8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global short8 *dst, const __local short8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ushort8 *dst, const __local ushort8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global int8 *dst, const __local int8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uint8 *dst, const __local uint8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global long8 *dst, const __local long8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ulong8 *dst, const __local ulong8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global float8 *dst, const __local float8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global char16 *dst, const __local char16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uchar16 *dst, const __local uchar16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global short16 *dst, const __local short16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ushort16 *dst, const __local ushort16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global int16 *dst, const __local int16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global uint16 *dst, const __local uint16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global long16 *dst, const __local long16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global ulong16 *dst, const __local ulong16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global float16 *dst, const __local float16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local double *dst, const __global double *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local double2 *dst, const __global double2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local double3 *dst, const __global double3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local double4 *dst, const __global double4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local double8 *dst, const __global double8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local double16 *dst, const __global double16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global double *dst, const __local double *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global double2 *dst, const __local double2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global double3 *dst, const __local double3 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global double4 *dst, const __local double4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global double8 *dst, const __local double8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global double16 *dst, const __local double16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local half *dst, const __global half *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local half2 *dst, const __global half2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local half4 *dst, const __global half4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local half8 *dst, const __global half8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__local half16 *dst, const __global half16 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global half *dst, const __local half *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global half2 *dst, const __local half2 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global half4 *dst, const __local half4 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global half8 *dst, const __local
```

```
 half8 *src, size_t num_elements, event_t event);
event_t __attribute__((overloadable)) async_work_group_copy(__global half16 *dst, const __local half16 *src, size_t num_elements, event_t event);

/**
 * Perform an async gather of num_elements
 * gentype elements from src to dst. The
 * src_stride is the stride in elements for each
 * gentype element read from src. The dst_stride
 * is the stride in elements for each gentype
 * element written to dst. The async gather is
 * performed by all work-items in a work-group.
 * This built-in function must therefore be
 * encountered by all work-items in a work-group
 * executing the kernel with the same argument
 * values; otherwise the results are undefined.
 * Returns an event object that can be used by
 * wait_group_events to wait for the async copy
 * to finish. The event argument can also be used
 * to associate the
 * async_work_group_strided_copy with a
 * previous async copy allowing an event to be
 * shared by multiple async copies; otherwise event
 * should be zero.
 * If event argument is non-zero, the event object
 * supplied in event argument will be returned.
 * This function does not perform any implicit
 * synchronization of source data such as using a
 * barrier before performing the copy.
 */
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local char *dst, const __global char *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uchar *dst, const __global uchar *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local short *dst, const __global short *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ushort *dst, const __global ushort *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local int *dst, const __global int *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uint *dst, const __global uint *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local long *dst, const __global long *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ulong *dst, const __global ulong *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local float *dst, const __global float *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local char2 *dst, const __global char2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uchar2 *dst, const __global uchar2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local short2 *dst, const __global short2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ushort2 *dst, const __global ushort2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local int2 *dst, const __global int2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uint2 *dst, const __global uint2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local long2 *dst, const __global long2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ulong2 *dst, const __global ulong2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local float2 *dst, const __global float2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local char3 *dst, const __global char3 *src, size_t num_elements, size_t src_stride, event_t event);
```

```
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uchar3 *dst, const __global uchar3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local short3 *dst, const __global short3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ushort3 *dst, const __global ushort3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local int3 *dst, const __global int3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uint3 *dst, const __global uint3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local long3 *dst, const __global long3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ulong3 *dst, const __global ulong3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local float3 *dst, const __global float3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local char4 *dst, const __global char4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uchar4 *dst, const __global uchar4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local short4 *dst, const __global short4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ushort4 *dst, const __global ushort4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local int4 *dst, const __global int4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uint4 *dst, const __global uint4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local long4 *dst, const __global long4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ulong4 *dst, const __global ulong4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local float4 *dst, const __global float4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local char8 *dst, const __global char8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uchar8 *dst, const __global uchar8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local short8 *dst, const __global short8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ushort8 *dst, const __global ushort8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local int8 *dst, const __global int8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uint8 *dst, const __global uint8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local long8 *dst, const __global long8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ulong8 *dst, const __global ulong8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local float8 *dst, const __global float8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local char16 *dst, const __global char16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uchar16 *dst, const __global uchar16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local short16 *dst, const __global short16 *src, size_t num_elements, size_t src_stride, event_t event);
```

```
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ushort16 *dst, const __global ushort16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local int16 *dst, const __global int16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local uint16 *dst, const __global uint16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local long16 *dst, const __global long16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local ulong16 *dst, const __global ulong16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local float16 *dst, const __global float16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global char *dst, const __local char *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uchar *dst, const __local uchar *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global short *dst, const __local short *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ushort *dst, const __local ushort *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global int *dst, const __local int *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uint *dst, const __local uint *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global long *dst, const __local long *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ulong *dst, const __local ulong *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global float *dst, const __local float *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global char2 *dst, const __local char2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uchar2 *dst, const __local uchar2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global short2 *dst, const __local short2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ushort2 *dst, const __local ushort2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global int2 *dst, const __local int2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uint2 *dst, const __local uint2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global long2 *dst, const __local long2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ulong2 *dst, const __local ulong2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global float2 *dst, const __local float2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global char3 *dst, const __local char3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uchar3 *dst, const __local uchar3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global short3 *dst, const __local short3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ushort3 *dst, const __local ushort3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global int3 *dst, const __local int3 *src, size_t num_elements, size_t dst_stride, event_t event);
```

```
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uint3 *dst, const __local uint3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global long3 *dst, const __local long3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ulong3 *dst, const __local ulong3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global float3 *dst, const __local float3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global char4 *dst, const __local char4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uchar4 *dst, const __local uchar4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global short4 *dst, const __local short4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ushort4 *dst, const __local ushort4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global int4 *dst, const __local int4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uint4 *dst, const __local uint4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global long4 *dst, const __local long4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ulong4 *dst, const __local ulong4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global float4 *dst, const __local float4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global char8 *dst, const __local char8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uchar8 *dst, const __local uchar8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global short8 *dst, const __local short8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ushort8 *dst, const __local ushort8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global int8 *dst, const __local int8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uint8 *dst, const __local uint8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global long8 *dst, const __local long8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ulong8 *dst, const __local ulong8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global float8 *dst, const __local float8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global char16 *dst, const __local char16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uchar16 *dst, const __local uchar16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global short16 *dst, const __local short16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ushort16 *dst, const __local ushort16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global int16 *dst, const __local int16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global uint16 *dst, const __local uint16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global long16 *dst, const __local long16 *src, size_t num_elements, size_t dst_stride, event_t event);
```

```
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global ulong16 *dst, const __local ulong16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global float16 *dst, const __local float16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local double *dst, const __global double *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local double2 *dst, const __global double2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local double3 *dst, const __global double3 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local double4 *dst, const __global double4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local double8 *dst, const __global double8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local double16 *dst, const __global double16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global double *dst, const __local double *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global double2 *dst, const __local double2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global double3 *dst, const __local double3 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global double4 *dst, const __local double4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global double8 *dst, const __local double8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global double16 *dst, const __local double16 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local half *dst, const __global half *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local half2 *dst, const __global half2 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local half4 *dst, const __global half4 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local half8 *dst, const __global half8 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__local half16 *dst, const __global half16 *src, size_t num_elements, size_t src_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global half *dst, const __local half *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global half2 *dst, const __local half2 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global half4 *dst, const __local half4 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global half8 *dst, const __local half8 *src, size_t num_elements, size_t dst_stride, event_t event);
event_t __attribute__((overloadable)) async_work_group_strided_copy(__global half16 *dst, const __local half16 *src, size_t num_elements, size_t dst_stride, event_t event);

/**
 * Wait for events that identify the
 * async_work_group_copy operations to
 * complete. The event objects specified in
 * event_list will be released after the wait is
 * performed.
 * This function must be encountered by all workitems
 * in a work-group executing the kernel with
 * the same num_events and event objects specified
 * in event_list; otherwise the results are undefined.
 */
void __attribute__((overloadable)) wait_group_events(int num_events, event_t * event_list);
```

```
/**
 * Prefetch num_elements * sizeof(gentype)
 * bytes into the global cache. The prefetch
 * instruction is applied to a work-item in a workgroup
 * and does not affect the functional
 * behavior of the kernel.
 */
void __attribute__((overloadable)) prefetch(const __global char *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uchar * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global short *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ushort * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global int *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uint * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global long *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ulong * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global float *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global char2 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uchar2 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global short2 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ushort2 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global int2 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uint2 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global long2 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ulong2 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global float2 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global char3 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uchar3 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global short3 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ushort3 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global int3 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uint3 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global long3 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ulong3 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global float3 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global char4 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uchar4 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global short4 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ushort4 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global int4 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uint4 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global long4 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ulong4 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global float4 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global char8 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uchar8 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global short8 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ushort8 * p, size_t num_elements);
```

```
void __attribute__((overloadable)) prefetch(const __global int8 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uint8 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global long8 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ulong8 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global float8 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global char16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uchar16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global short16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ushort16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global int16 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global uint16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global long16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global ulong16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global float16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global double * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global double2 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global double3 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global double4 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global double8 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global double16 * p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global half *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global half2 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global half4 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global half8 *p, size_t num_elements);
void __attribute__((overloadable)) prefetch(const __global half16 * p, size_t num_elements);

// Atomic functions

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old + val) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_add(volatile __global int * p, int val);
unsigned int __attribute__((overloadable)) atomic_add(volatile __global unsigned int * p, unsigned int val);
int __attribute__((overloadable)) atomic_add(volatile __local int * p, int val);
unsigned int __attribute__((overloadable)) atomic_add(volatile __local unsigned int * p, unsigned int val);

int __attribute__((overloadable)) atom_add(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_add(__global unsigned int * p, unsigned int val);
int __attribute__((overloadable)) atom_add(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_add(__local unsigned int * p, unsigned int val);

int __attribute__((overloadable)) atom_add(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_add(volatile __global unsigned int * p, unsigned int val);
int __attribute__((overloadable)) atom_add(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_add(volatile __local unsigned int * p, unsigned int val);

/**
 * unsigned int atomic_sub (
```

```
 * volatile __global unsigned int *p,
 * unsigned int val)
 * int atomic_sub (volatile __local int *p, int val)
 * unsigned int atomic_sub (
 * volatile __local unsigned int *p,
 * unsigned int val)
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old - val) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_sub(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atomic_sub(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atomic_sub(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_sub(volatile __local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_sub(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_sub(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_sub(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_sub(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_sub(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_sub(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_sub(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_sub(volatile __local unsigned int *p, unsigned int val);

/**
 * Swaps the old value stored at location p
 * with new value given by val. Returns old
 * value.
 */
int __attribute__((overloadable)) atomic_xchg(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atomic_xchg(volatile __global unsigned int *p, unsigned int val);
float __attribute__((overloadable)) atomic_xchg(volatile __global float *p, float val);
int __attribute__((overloadable)) atomic_xchg(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_xchg(volatile __local unsigned int *p, unsigned int val);
float __attribute__((overloadable)) atomic_xchg(volatile __local float *p, float val);

int __attribute__((overloadable)) atom_xchg(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_xchg(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_xchg(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_xchg(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_xchg(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_xchg(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_xchg(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_xchg(volatile __local unsigned int *p, unsigned int val);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old + 1) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_inc(volatile __global int *p);
unsigned int __attribute__((overloadable)) atomic_inc(volatile __global unsigned int *p);
int __attribute__((overloadable)) atomic_inc(volatile __local int *p);
unsigned int __attribute__((overloadable)) atomic_inc(volatile __local unsigned int *p);
```

```

int __attribute__((overloadable)) atom_inc(__global int *p);
unsigned int __attribute__((overloadable)) atom_inc(__global unsigned int *p);
int __attribute__((overloadable)) atom_inc(__local int *p);
unsigned int __attribute__((overloadable)) atom_inc(__local unsigned int *p);

int __attribute__((overloadable)) atom_inc(volatile __global int *p);
unsigned int __attribute__((overloadable)) atom_inc(volatile __global unsigned int *p);
int __attribute__((overloadable)) atom_inc(volatile __local int *p);
unsigned int __attribute__((overloadable)) atom_inc(volatile __local unsigned int *p);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old - 1) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_dec(volatile __global int *p);
unsigned int __attribute__((overloadable)) atomic_dec(volatile __global unsigned int *p);
int __attribute__((overloadable)) atomic_dec(volatile __local int *p);
unsigned int __attribute__((overloadable)) atomic_dec(volatile __local unsigned int *p);

int __attribute__((overloadable)) atom_dec(__global int *p);
unsigned int __attribute__((overloadable)) atom_dec(__global unsigned int *p);
int __attribute__((overloadable)) atom_dec(__local int *p);
unsigned int __attribute__((overloadable)) atom_dec(__local unsigned int *p);

int __attribute__((overloadable)) atom_dec(volatile __global int *p);
unsigned int __attribute__((overloadable)) atom_dec(volatile __global unsigned int *p);
int __attribute__((overloadable)) atom_dec(volatile __local int *p);
unsigned int __attribute__((overloadable)) atom_dec(volatile __local unsigned int *p);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old == cmp) ? val : old and store result at
 * location pointed by p. The function
 * returns old.
 */
int __attribute__((overloadable)) atomic_cmpxchg(volatile __global int *p, int cmp, int val);
unsigned int __attribute__((overloadable)) atomic_cmpxchg(volatile __global unsigned int *p, unsigned int cmp, unsigned int val);
int __attribute__((overloadable)) atomic_cmpxchg(volatile __local int *p, int cmp, int val);
unsigned int __attribute__((overloadable)) atomic_cmpxchg(volatile __local unsigned int *p, unsigned int cmp, unsigned int val);

int __attribute__((overloadable)) atom_cmpxchg(__global int *p, int cmp, int val);
unsigned int __attribute__((overloadable)) atom_cmpxchg(__global unsigned int *p, unsigned int cmp, unsigned int val);
int __attribute__((overloadable)) atom_cmpxchg(__local int *p, int cmp, int val);
unsigned int __attribute__((overloadable)) atom_cmpxchg(__local unsigned int *p, unsigned int cmp, unsigned int val);

int __attribute__((overloadable)) atom_cmpxchg(volatile __global int *p, int cmp, int val);
unsigned int __attribute__((overloadable)) atom_cmpxchg(volatile __global unsigned int *p, unsigned int cmp, unsigned int val);
int __attribute__((overloadable)) atom_cmpxchg(volatile __local int *p, int cmp, int val);
unsigned int __attribute__((overloadable)) atom_cmpxchg(volatile __local unsigned int *p, unsigned int cmp, unsigned int val);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * min(old, val) and store minimum value at
 * location pointed by p. The function
 * returns old.
 */
int __attribute__((overloadable)) atomic_min(volatile __global int *p, int val);
```

```
unsigned int __attribute__((overloadable)) atomic_min(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atomic_min(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_min(volatile __local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_min(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_min(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_min(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_min(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_min(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_min(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_min(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_min(volatile __local unsigned int *p, unsigned int val);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * max(old, val) and store maximum value at
 * location pointed by p. The function
 * returns old.
 */
int __attribute__((overloadable)) atomic_max(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atomic_max(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atomic_max(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_max(volatile __local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_max(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_max(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_max(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_max(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_max(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_max(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_max(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_max(volatile __local unsigned int *p, unsigned int val);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old & val) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_and(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atomic_and(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atomic_and(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_and(volatile __local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_and(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_and(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_and(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_and(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_and(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_and(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_and(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_and(volatile __local unsigned int *
```

```
p, unsigned int val);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old | val) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_or(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atomic_or(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atomic_or(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_or(volatile __local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_or(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_or(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_or(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_or(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_or(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_or(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_or(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_or(volatile __local unsigned int *p, unsigned int val);

/**
 * Read the 32-bit value (referred to as old)
 * stored at location pointed by p. Compute
 * (old ^ val) and store result at location
 * pointed by p. The function returns old.
 */
int __attribute__((overloadable)) atomic_xor(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atomic_xor(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atomic_xor(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atomic_xor(volatile __local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_xor(__global int *p, int val);
unsigned int __attribute__((overloadable)) atom_xor(__global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_xor(__local int *p, int val);
unsigned int __attribute__((overloadable)) atom_xor(__local unsigned int *p, unsigned int val);

int __attribute__((overloadable)) atom_xor(volatile __global int *p, int val);
unsigned int __attribute__((overloadable)) atom_xor(volatile __global unsigned int *p, unsigned int val);
int __attribute__((overloadable)) atom_xor(volatile __local int *p, int val);
unsigned int __attribute__((overloadable)) atom_xor(volatile __local unsigned int *p, unsigned int val);

// Function qualifiers (section 6.7)

#define __kernel_exec(X, typen) __kernel \
    __attribute__((work_group_size_hint(X, 1, 1))) \
    __attribute__((vec_type_hint(typen)))

#define kernel_exec(X, typen) __kernel \
    __attribute__((work_group_size_hint(X, 1, 1))) \
    __attribute__((vec_type_hint(typen)))

// Miscellaneous vector functions

/**
 * The shuffle and shuffle2 built-in functions construct
 * a permutation of elements from one or two input
 * vectors respectively that are of the same type,
 * returning a vector with the same element type as the
 * input and length that is the same as the shuffle mask.
 * The size of each element in the mask must match the
 * size of each element in the result. For shuffle, only
 * the ilogb(2m-1) least significant bits of each mask
 * element are considered. For shuffle2, only the
```

```
 * ilogb(2m-1)+1 least significant bits of each mask
 * element are considered. Other bits in the mask shall
 * be ignored.
 * The elements of the input vectors are numbered from
 * left to right across one or both of the vectors. For this
 * purpose, the number of elements in a vector is given
 * by vec_step(gentypem). The shuffle mask operand
 * specifies, for each element of the result vector, which
 * element of the one or two input vectors the result
 * element gets.
 * Examples:
 * uint4 mask = (uint4)(3, 2,
 * 1, 0);
 * float4 a;
 * float4 r = shuffle(a, mask);
 * // r.s0123 = a.wzyx
 * uint8 mask = (uint8)(0, 1, 2, 3,
 * 4, 5, 6, 7);
 * float4 a, b;
 * float8 r = shuffle2(a, b, mask);
 * // r.s0123 = a.xyzw
 * // r.s4567 = b.xyzw
 * uint4 mask;
 * float8 a;
 * float4 b;
 * b = shuffle(a, mask);
 * Examples that are not valid are:
 * uint8 mask;
 * short16 a;
 * short8 b;
 * b = shuffle(a, mask); <- not valid
 */
char2       const_func __attribute__((overloadable)) shuffle(char2 x, uchar2 mask);
char2       const_func __attribute__((overloadable)) shuffle(char4 x, uchar2 mask);
char2       const_func __attribute__((overloadable)) shuffle(char8 x, uchar2 mask);
char2       const_func __attribute__((overloadable)) shuffle(char16 x, uchar2 mask);

uchar2      const_func __attribute__((overloadable)) shuffle(uchar2 x, uchar2 mask);
uchar2      const_func __attribute__((overloadable)) shuffle(uchar4 x, uchar2 mask);
uchar2      const_func __attribute__((overloadable)) shuffle(uchar8 x, uchar2 mask);
uchar2      const_func __attribute__((overloadable)) shuffle(uchar16 x, uchar2 mask);

short2 const_func __attribute__((overloadable)) shuffle(short2 x, ushort2 mask);
short2 const_func __attribute__((overloadable)) shuffle(short4 x, ushort2 mask);
short2 const_func __attribute__((overloadable)) shuffle(short8 x, ushort2 mask);
short2 const_func __attribute__((overloadable)) shuffle(short16 x, ushort2 mask);

ushort2 const_func __attribute__((overloadable)) shuffle(ushort2 x, ushort2 mask);
ushort2 const_func __attribute__((overloadable)) shuffle(ushort4 x, ushort2 mask);
ushort2 const_func __attribute__((overloadable)) shuffle(ushort8 x, ushort2 mask);
ushort2 const_func __attribute__((overloadable)) shuffle(ushort16 x, ushort2 mask);

int2 const_func __attribute__((overloadable)) shuffle(int2 x, uint2 mask);
int2 const_func __attribute__((overloadable)) shuffle(int4 x, uint2 mask);
int2 const_func __attribute__((overloadable)) shuffle(int8 x, uint2 mask);
int2 const_func __attribute__((overloadable)) shuffle(int16 x, uint2 mask);

uint2 const_func __attribute__((overloadable)) shuffle(uint2 x, uint2 mask);
uint2 const_func __attribute__((overloadable)) shuffle(uint4 x, uint2 mask);
uint2 const_func __attribute__((overloadable)) shuffle(uint8 x, uint2 mask);
uint2 const_func __attribute__((overloadable)) shuffle(uint16 x, uint2 mask);

long2 const_func __attribute__((overloadable)) shuffle(long2
```

```
x, ulong2 mask);
long2 const_func __attribute__((overloadable)) shuffle(long4
x, ulong2 mask);
long2 const_func __attribute__((overloadable)) shuffle(long8
x, ulong2 mask);
long2 const_func __attribute__((overloadable)) shuffle(long16
x, ulong2 mask);

ulong2 const_func __attribute__((overloadable)) shuffle(ulong2
x, ulong2 mask);
ulong2 const_func __attribute__((overloadable)) shuffle(ulong4
x, ulong2 mask);
ulong2 const_func __attribute__((overloadable)) shuffle(ulong8
x, ulong2 mask);
ulong2 const_func __attribute__((overloadable)) shuffle(ulong16
x, ulong2 mask);

//half2 const_func __attribute__((overloadable)) shuffle(half2 x, ushort2 mask);
//half2 const_func __attribute__((overloadable)) shuffle(half4 x, ushort2 mask);
//half2 const_func __attribute__((overloadable)) shuffle(half8 x, ushort2 mask);
//half2 const_func __attribute__((overloadable)) shuffle(half16 x, ushort2 mask);

float2 const_func __attribute__((overloadable)) shuffle(float2
x, uint2 mask);
float2 const_func __attribute__((overloadable)) shuffle(float4
x, uint2 mask);
float2 const_func __attribute__((overloadable)) shuffle(float8
x, uint2 mask);
float2 const_func __attribute__((overloadable)) shuffle(float16
x, uint2 mask);

double2 const_func __attribute__((overloadable)) shuffle(double2
x, ulong2 mask);
double2 const_func __attribute__((overloadable)) shuffle(double4
x, ulong2 mask);
double2 const_func __attribute__((overloadable)) shuffle(double8
x, ulong2 mask);
double2 const_func __attribute__((overloadable)) shuffle(double16
x, ulong2 mask);

char4 const_func __attribute__((overloadable)) shuffle(char2
x, uchar4 mask);
char4 const_func __attribute__((overloadable)) shuffle(char4
x, uchar4 mask);
char4 const_func __attribute__((overloadable)) shuffle(char8
x, uchar4 mask);
char4 const_func __attribute__((overloadable)) shuffle(char16
x, uchar4 mask);

uchar4 const_func __attribute__((overloadable)) shuffle(uchar2
x, uchar4 mask);
uchar4 const_func __attribute__((overloadable)) shuffle(uchar4
x, uchar4 mask);
uchar4 const_func __attribute__((overloadable)) shuffle(uchar8
x, uchar4 mask);
uchar4 const_func __attribute__((overloadable)) shuffle(uchar16
x, uchar4 mask);

short4 const_func __attribute__((overloadable)) shuffle(short2
x, ushort4 mask);
short4 const_func __attribute__((overloadable)) shuffle(short4
x, ushort4 mask);
short4 const_func __attribute__((overloadable)) shuffle(short8
x, ushort4 mask);
short4 const_func __attribute__((overloadable)) shuffle(short16
x, ushort4 mask);

ushort4 const_func __attribute__((overloadable)) shuffle(ushort2
x, ushort4 mask);
ushort4 const_func __attribute__((overloadable)) shuffle(ushort4
x, ushort4 mask);
ushort4 const_func __attribute__((overloadable)) shuffle(ushort8
x, ushort4 mask);
ushort4 const_func __attribute__((overloadable)) shuffle(ushort16
x, ushort4 mask);

int4 const_func __attribute__((overloadable)) shuffle(int2
x, uint4 mask);
int4 const_func __attribute__((overloadable)) shuffle(int4
x, uint4 mask);
int4 const_func __attribute__((overloadable)) shuffle(int8
x, uint4 mask);
int4 const_func __attribute__((overloadable)) shuffle(int16
x, uint4 mask);

uint4 const_func __attribute__((overloadable)) shuffle(uint2
x, uint4 mask);
```

```
uint4 const_func __attribute__((overloadable)) shuffle(uint4 x, uint4 mask);
uint4 const_func __attribute__((overloadable)) shuffle(uint8 x, uint4 mask);
uint4 const_func __attribute__((overloadable)) shuffle(uint16 x, uint4 mask);

long4 const_func __attribute__((overloadable)) shuffle(long2 x, ulong4 mask);
long4 const_func __attribute__((overloadable)) shuffle(long4 x, ulong4 mask);
long4 const_func __attribute__((overloadable)) shuffle(long8 x, ulong4 mask);
long4 const_func __attribute__((overloadable)) shuffle(long16 x, ulong4 mask);

ulong4 const_func __attribute__((overloadable)) shuffle(ulong2 x, ulong4 mask);
ulong4 const_func __attribute__((overloadable)) shuffle(ulong4 x, ulong4 mask);
ulong4 const_func __attribute__((overloadable)) shuffle(ulong8 x, ulong4 mask);
ulong4 const_func __attribute__((overloadable)) shuffle(ulong16 x, ulong4 mask);

//half4 const_func __attribute__((overloadable)) shuffle(half2 x, ushort4 mask);
//half4 const_func __attribute__((overloadable)) shuffle(half4 x, ushort4 mask);
//half4 const_func __attribute__((overloadable)) shuffle(half8 x, ushort4 mask);
//half4 const_func __attribute__((overloadable)) shuffle(half16 x, ushort4 mask);

float4 const_func __attribute__((overloadable)) shuffle(float2 x, uint4 mask);
float4 const_func __attribute__((overloadable)) shuffle(float4 x, uint4 mask);
float4 const_func __attribute__((overloadable)) shuffle(float8 x, uint4 mask);
float4 const_func __attribute__((overloadable)) shuffle(float16 x, uint4 mask);

double4 const_func __attribute__((overloadable)) shuffle(double2 x, ulong4 mask);
double4 const_func __attribute__((overloadable)) shuffle(double4 x, ulong4 mask);
double4 const_func __attribute__((overloadable)) shuffle(double8 x, ulong4 mask);
double4 const_func __attribute__((overloadable)) shuffle(double16 x, ulong4 mask);

char8 const_func __attribute__((overloadable)) shuffle(char2 x, uchar8 mask);
char8 const_func __attribute__((overloadable)) shuffle(char4 x, uchar8 mask);
char8 const_func __attribute__((overloadable)) shuffle(char8 x, uchar8 mask);
char8 const_func __attribute__((overloadable)) shuffle(char16 x, uchar8 mask);

uchar8 const_func __attribute__((overloadable)) shuffle(uchar2 x, uchar8 mask);
uchar8 const_func __attribute__((overloadable)) shuffle(uchar4 x, uchar8 mask);
uchar8 const_func __attribute__((overloadable)) shuffle(uchar8 x, uchar8 mask);
uchar8 const_func __attribute__((overloadable)) shuffle(uchar16 x, uchar8 mask);

short8 const_func __attribute__((overloadable)) shuffle(short2 x, ushort8 mask);
short8 const_func __attribute__((overloadable)) shuffle(short4 x, ushort8 mask);
short8 const_func __attribute__((overloadable)) shuffle(short8 x, ushort8 mask);
short8 const_func __attribute__((overloadable)) shuffle(short16 x, ushort8 mask);

ushort8 const_func __attribute__((overloadable)) shuffle(ushort2 x, ushort8 mask);
ushort8 const_func __attribute__((overloadable)) shuffle(ushort4 x, ushort8 mask);
ushort8 const_func __attribute__((overloadable)) shuffle(ushort8 x, ushort8 mask);
ushort8 const_func __attribute__((overloadable)) shuffle(ushort16 x, ushort8 mask);

int8 const_func __attribute__((overloadable)) shuffle(int2 x, uint8 mask);
int8 const_func __attribute__((overloadable)) shuffle(int4
```

```
x, uint8 mask);
int8 const_func __attribute__((overloadable)) shuffle(int8
x, uint8 mask);
int8 const_func __attribute__((overloadable)) shuffle(int16
x, uint8 mask);

uint8 const_func __attribute__((overloadable)) shuffle(uint2
x, uint8 mask);
uint8 const_func __attribute__((overloadable)) shuffle(uint4
x, uint8 mask);
uint8 const_func __attribute__((overloadable)) shuffle(uint8
x, uint8 mask);
uint8 const_func __attribute__((overloadable)) shuffle(uint16
x, uint8 mask);

long8 const_func __attribute__((overloadable)) shuffle(long2
x, ulong8 mask);
long8 const_func __attribute__((overloadable)) shuffle(long4
x, ulong8 mask);
long8 const_func __attribute__((overloadable)) shuffle(long8
x, ulong8 mask);
long8 const_func __attribute__((overloadable)) shuffle(long16
x, ulong8 mask);

ulong8 const_func __attribute__((overloadable)) shuffle(ulong2
x, ulong8 mask);
ulong8 const_func __attribute__((overloadable)) shuffle(ulong4
x, ulong8 mask);
ulong8 const_func __attribute__((overloadable)) shuffle(ulong8
x, ulong8 mask);
ulong8 const_func __attribute__((overloadable)) shuffle(ulong16
x, ulong8 mask);

//half8 const_func __attribute__((overloadable)) shuffle(half2 x, ushort8 mask);
//half8 const_func __attribute__((overloadable)) shuffle(half4 x, ushort8 mask);
//half8 const_func __attribute__((overloadable)) shuffle(half8 x, ushort8 mask);
//half8 const_func __attribute__((overloadable)) shuffle(half16 x, ushort8 mask);

float8 const_func __attribute__((overloadable)) shuffle(float2
x, uint8 mask);
float8 const_func __attribute__((overloadable)) shuffle(float4
x, uint8 mask);
float8 const_func __attribute__((overloadable)) shuffle(float8
x, uint8 mask);
float8 const_func __attribute__((overloadable)) shuffle(float16
x, uint8 mask);

double8 const_func __attribute__((overloadable)) shuffle(double2
x, ulong8 mask);
double8 const_func __attribute__((overloadable)) shuffle(double4
x, ulong8 mask);
double8 const_func __attribute__((overloadable)) shuffle(double8
x, ulong8 mask);
double8 const_func __attribute__((overloadable)) shuffle(double16
x, ulong8 mask);

char16 const_func __attribute__((overloadable)) shuffle(char2
x, uchar16 mask);
char16 const_func __attribute__((overloadable)) shuffle(char4
x, uchar16 mask);
char16 const_func __attribute__((overloadable)) shuffle(char8
x, uchar16 mask);
char16 const_func __attribute__((overloadable)) shuffle(char16
x, uchar16 mask);

uchar16 const_func __attribute__((overloadable)) shuffle(uchar2
x, uchar16 mask);
uchar16 const_func __attribute__((overloadable)) shuffle(uchar4
x, uchar16 mask);
uchar16 const_func __attribute__((overloadable)) shuffle(uchar8
x, uchar16 mask);
uchar16 const_func __attribute__((overloadable)) shuffle(uchar16
x, uchar16 mask);

short16 const_func __attribute__((overloadable)) shuffle(short2
x, ushort16 mask);
short16 const_func __attribute__((overloadable)) shuffle(short4
x, ushort16 mask);
short16 const_func __attribute__((overloadable)) shuffle(short8
x, ushort16 mask);
short16 const_func __attribute__((overloadable)) shuffle(short16
x, ushort16 mask);

ushort16 const_func __attribute__((overloadable)) shuffle(ushort2
x, ushort16 mask);
ushort16 const_func __attribute__((overloadable)) shuffle(ushort4
x, ushort16 mask);
```

```
ushort16 const_func __attribute__((overloadable)) shuffle(ushort8 x, ushort16 mask);
ushort16 const_func __attribute__((overloadable)) shuffle(ushort16 x, ushort16 mask);

int16 const_func __attribute__((overloadable)) shuffle(int2 x, uint16 mask);
int16 const_func __attribute__((overloadable)) shuffle(int4 x, uint16 mask);
int16 const_func __attribute__((overloadable)) shuffle(int8 x, uint16 mask);
int16 const_func __attribute__((overloadable)) shuffle(int16 x, uint16 mask);

uint16 const_func __attribute__((overloadable)) shuffle(uint2 x, uint16 mask);
uint16 const_func __attribute__((overloadable)) shuffle(uint4 x, uint16 mask);
uint16 const_func __attribute__((overloadable)) shuffle(uint8 x, uint16 mask);
uint16 const_func __attribute__((overloadable)) shuffle(uint16 x, uint16 mask);

long16 const_func __attribute__((overloadable)) shuffle(long2 x, ulong16 mask);
long16 const_func __attribute__((overloadable)) shuffle(long4 x, ulong16 mask);
long16 const_func __attribute__((overloadable)) shuffle(long8 x, ulong16 mask);
long16 const_func __attribute__((overloadable)) shuffle(long16 x, ulong16 mask);

ulong16 const_func __attribute__((overloadable)) shuffle(ulong2 x, ulong16 mask);
ulong16 const_func __attribute__((overloadable)) shuffle(ulong4 x, ulong16 mask);
ulong16 const_func __attribute__((overloadable)) shuffle(ulong8 x, ulong16 mask);
ulong16 const_func __attribute__((overloadable)) shuffle(ulong16 x, ulong16 mask);

//half16 const_func __attribute__((overloadable)) shuffle(half2 x, ushort16 mask);
//half16 const_func __attribute__((overloadable)) shuffle(half4 x, ushort16 mask);
//half16 const_func __attribute__((overloadable)) shuffle(half8 x, ushort16 mask);
//half16 const_func __attribute__((overloadable)) shuffle(half16 x, ushort16 mask);

float16 const_func __attribute__((overloadable)) shuffle(float2 x, uint16 mask);
float16 const_func __attribute__((overloadable)) shuffle(float4 x, uint16 mask);
float16 const_func __attribute__((overloadable)) shuffle(float8 x, uint16 mask);
float16 const_func __attribute__((overloadable)) shuffle(float16 x, uint16 mask);

double16 const_func __attribute__((overloadable)) shuffle(double2 x, ulong16 mask);
double16 const_func __attribute__((overloadable)) shuffle(double4 x, ulong16 mask);
double16 const_func __attribute__((overloadable)) shuffle(double8 x, ulong16 mask);
double16 const_func __attribute__((overloadable)) shuffle(double16 x, ulong16 mask);


char2 const_func __attribute__((overloadable)) shuffle2(char2 x, char2 y, uchar2 mask);
char2 const_func __attribute__((overloadable)) shuffle2(char4 x, char4 y, uchar2 mask);
char2 const_func __attribute__((overloadable)) shuffle2(char8 x, char8 y, uchar2 mask);
char2 const_func __attribute__((overloadable)) shuffle2(char16 x, char16 y, uchar2 mask);

uchar2 const_func __attribute__((overloadable)) shuffle2(uchar2 x, uchar2 y, uchar2 mask);
uchar2 const_func __attribute__((overloadable)) shuffle2(uchar4 x, uchar4 y, uchar2 mask);
uchar2 const_func __attribute__((overloadable)) shuffle2(uchar8 x, uchar8 y, uchar2 mask);
uchar2 const_func __attribute__((overloadable)) shuffle2(uchar16 x, uchar16 y, uchar2 mask);

short2 const_func __attribute__((overloadable)) shuffle2(short2 x, short2 y, ushort2 mask);
short2 const_func __attribute__((overloadable)) shuffle2(short4 x, short4 y, ushort2 mask);
```

```
short2 const_func __attribute__((overloadable)) shuffle2(short8 x, short8 y, ushort2 mask);
short2 const_func __attribute__((overloadable)) shuffle2(short16 x, short16 y, ushort2 mask);

ushort2 const_func __attribute__((overloadable)) shuffle2(ushort2 x, ushort2 y, ushort2 mask);
ushort2 const_func __attribute__((overloadable)) shuffle2(ushort4 x, ushort4 y, ushort2 mask);
ushort2 const_func __attribute__((overloadable)) shuffle2(ushort8 x, ushort8 y, ushort2 mask);
ushort2 const_func __attribute__((overloadable)) shuffle2(ushort16 x, ushort16 y, ushort2 mask);

int2 const_func __attribute__((overloadable)) shuffle2(int2 x, int2 y, uint2 mask);
int2 const_func __attribute__((overloadable)) shuffle2(int4 x, int4 y, uint2 mask);
int2 const_func __attribute__((overloadable)) shuffle2(int8 x, int8 y, uint2 mask);
int2 const_func __attribute__((overloadable)) shuffle2(int16 x, int16 y, uint2 mask);

uint2 const_func __attribute__((overloadable)) shuffle2(uint2 x, uint2 y, uint2 mask);
uint2 const_func __attribute__((overloadable)) shuffle2(uint4 x, uint4 y, uint2 mask);
uint2 const_func __attribute__((overloadable)) shuffle2(uint8 x, uint8 y, uint2 mask);
uint2 const_func __attribute__((overloadable)) shuffle2(uint16 x, uint16 y, uint2 mask);

long2 const_func __attribute__((overloadable)) shuffle2(long2 x, long2 y, ulong2 mask);
long2 const_func __attribute__((overloadable)) shuffle2(long4 x, long4 y, ulong2 mask);
long2 const_func __attribute__((overloadable)) shuffle2(long8 x, long8 y, ulong2 mask);
long2 const_func __attribute__((overloadable)) shuffle2(long16 x, long16 y, ulong2 mask);

ulong2 const_func __attribute__((overloadable)) shuffle2(ulong2 x, ulong2 y, ulong2 mask);
ulong2 const_func __attribute__((overloadable)) shuffle2(ulong4 x, ulong4 y, ulong2 mask);
ulong2 const_func __attribute__((overloadable)) shuffle2(ulong8 x, ulong8 y, ulong2 mask);
ulong2 const_func __attribute__((overloadable)) shuffle2(ulong16 x, ulong16 y, ulong2 mask);

//half2 const_func __attribute__((overloadable)) shuffle2(half2 x, half2 y, ushort2 mask);
//half2 const_func __attribute__((overloadable)) shuffle2(half4 x, half4 y, ushort2 mask);
//half2 const_func __attribute__((overloadable)) shuffle2(half8 x, half8 y, ushort2 mask);
//half2 const_func __attribute__((overloadable)) shuffle2(half16 x, half16 y, ushort2 mask);

float2 const_func __attribute__((overloadable)) shuffle2(float2 x, float2 y, uint2 mask);
float2 const_func __attribute__((overloadable)) shuffle2(float4 x, float4 y, uint2 mask);
float2 const_func __attribute__((overloadable)) shuffle2(float8 x, float8 y, uint2 mask);
float2 const_func __attribute__((overloadable)) shuffle2(float16 x, float16 y, uint2 mask);

double2 const_func __attribute__((overloadable)) shuffle2(double2 x, double2 y, ulong2 mask);
double2 const_func __attribute__((overloadable)) shuffle2(double4 x, double4 y, ulong2 mask);
double2 const_func __attribute__((overloadable)) shuffle2(double8 x, double8 y, ulong2 mask);
double2 const_func __attribute__((overloadable)) shuffle2(double16 x, double16 y, ulong2 mask);

char4 const_func __attribute__((overloadable)) shuffle2(char2 x, char2 y, uchar4 mask);
char4 const_func __attribute__((overloadable)) shuffle2(char4 x, char4 y, uchar4 mask);
char4 const_func __attribute__((overloadable)) shuffle2(char8 x, char8 y, uchar4 mask);
char4 const_func __attribute__((overloadable)) shuffle2(char16 x, char16 y, uchar4 mask);

uchar4 const_func __attribute__((overloadable)) shuffle2(uchar2 x, uchar2 y, uchar4 mask);
uchar4 const_func __attribute__((overloadable)) shuffle2(uchar4 x, uchar4 y, uchar4 mask);
uchar4 const_func __attribute__((overloadable)) shuffle2(uchar8
```

```
x, uchar8 y, uchar4 mask);
uchar4 const_func __attribute__((overloadable)) shuffle2(uchar16
x, uchar16 y, uchar4 mask);

short4 const_func __attribute__((overloadable)) shuffle2(short2
x, short2 y, ushort4 mask);
short4 const_func __attribute__((overloadable)) shuffle2(short4
x, short4 y, ushort4 mask);
short4 const_func __attribute__((overloadable)) shuffle2(short8
x, short8 y, ushort4 mask);
short4 const_func __attribute__((overloadable)) shuffle2(short16
x, short16 y, ushort4 mask);

ushort4 const_func __attribute__((overloadable)) shuffle2(ushort2
x, ushort2 y, ushort4 mask);
ushort4 const_func __attribute__((overloadable)) shuffle2(ushort4
x, ushort4 y, ushort4 mask);
ushort4 const_func __attribute__((overloadable)) shuffle2(ushort8
x, ushort8 y, ushort4 mask);
ushort4 const_func __attribute__((overloadable)) shuffle2(ushort16
x, ushort16 y, ushort4 mask);

int4 const_func __attribute__((overloadable)) shuffle2(int2
x, int2 y, uint4 mask);
int4 const_func __attribute__((overloadable)) shuffle2(int4
x, int4 y, uint4 mask);
int4 const_func __attribute__((overloadable)) shuffle2(int8
x, int8 y, uint4 mask);
int4 const_func __attribute__((overloadable)) shuffle2(int16
x, int16 y, uint4 mask);

uint4 const_func __attribute__((overloadable)) shuffle2(uint2
x, uint2 y, uint4 mask);
uint4 const_func __attribute__((overloadable)) shuffle2(uint4
x, uint4 y, uint4 mask);
uint4 const_func __attribute__((overloadable)) shuffle2(uint8
x, uint8 y, uint4 mask);
uint4 const_func __attribute__((overloadable)) shuffle2(uint16
x, uint16 y, uint4 mask);

long4 const_func __attribute__((overloadable)) shuffle2(long2
x, long2 y, ulong4 mask);
long4 const_func __attribute__((overloadable)) shuffle2(long4
x, long4 y, ulong4 mask);
long4 const_func __attribute__((overloadable)) shuffle2(long8
x, long8 y, ulong4 mask);
long4 const_func __attribute__((overloadable)) shuffle2(long16
x, long16 y, ulong4 mask);

ulong4 const_func __attribute__((overloadable)) shuffle2(ulong2
x, ulong2 y, ulong4 mask);
ulong4 const_func __attribute__((overloadable)) shuffle2(ulong4
x, ulong4 y, ulong4 mask);
ulong4 const_func __attribute__((overloadable)) shuffle2(ulong8
x, ulong8 y, ulong4 mask);
ulong4 const_func __attribute__((overloadable)) shuffle2(ulong16
x, ulong16 y, ulong4 mask);

//half4 const_func __attribute__((overloadable)) shuffle2(half2 x, half2 y, ushort4 mask);
//half4 const_func __attribute__((overloadable)) shuffle2(half4 x, half4 y, ushort4 mask);
//half4 const_func __attribute__((overloadable)) shuffle2(half8 x, half8 y, ushort4 mask);
//half4 const_func __attribute__((overloadable)) shuffle2(half16 x, half16 y, ushort4 mask);

float4 const_func __attribute__((overloadable)) shuffle2(float2
x, float2 y, uint4 mask);
float4 const_func __attribute__((overloadable)) shuffle2(float4
x, float4 y, uint4 mask);
float4 const_func __attribute__((overloadable)) shuffle2(float8
x, float8 y, uint4 mask);
float4 const_func __attribute__((overloadable)) shuffle2(float16
x, float16 y, uint4 mask);

double4 const_func __attribute__((overloadable)) shuffle2(double2
x, double2 y, ulong4 mask);
double4 const_func __attribute__((overloadable)) shuffle2(double4
x, double4 y, ulong4 mask);
double4 const_func __attribute__((overloadable)) shuffle2(double8
x, double8 y, ulong4 mask);
double4 const_func __attribute__((overloadable)) shuffle2(double16
x, double16 y, ulong4 mask);

char8 const_func __attribute__((overloadable)) shuffle2(char2
x, char2 y, uchar8 mask);
char8 const_func __attribute__((overloadable)) shuffle2(char4
x, char4 y, uchar8 mask);
char8 const_func __attribute__((overloadable)) shuffle2(char8
x, char8 y, uchar8 mask);
```

```
char8 const_func __attribute__((overloadable)) shuffle2(char16 x, char16 y, uchar8 mask);

uchar8 const_func __attribute__((overloadable)) shuffle2(uchar2 x, uchar2 y, uchar8 mask);
uchar8 const_func __attribute__((overloadable)) shuffle2(uchar4 x, uchar4 y, uchar8 mask);
uchar8 const_func __attribute__((overloadable)) shuffle2(uchar8 x, uchar8 y, uchar8 mask);
uchar8 const_func __attribute__((overloadable)) shuffle2(uchar16 x, uchar16 y, uchar8 mask);

short8 const_func __attribute__((overloadable)) shuffle2(short2 x, short2 y, ushort8 mask);
short8 const_func __attribute__((overloadable)) shuffle2(short4 x, short4 y, ushort8 mask);
short8 const_func __attribute__((overloadable)) shuffle2(short8 x, short8 y, ushort8 mask);
short8 const_func __attribute__((overloadable)) shuffle2(short16 x, short16 y, ushort8 mask);

ushort8 const_func __attribute__((overloadable)) shuffle2(ushort2 x, ushort2 y, ushort8 mask);
ushort8 const_func __attribute__((overloadable)) shuffle2(ushort4 x, ushort4 y, ushort8 mask);
ushort8 const_func __attribute__((overloadable)) shuffle2(ushort8 x, ushort8 y, ushort8 mask);
ushort8 const_func __attribute__((overloadable)) shuffle2(ushort16 x, ushort16 y, ushort8 mask);

int8 const_func __attribute__((overloadable)) shuffle2(int2 x, int2 y, uint8 mask);
int8 const_func __attribute__((overloadable)) shuffle2(int4 x, int4 y, uint8 mask);
int8 const_func __attribute__((overloadable)) shuffle2(int8 x, int8 y, uint8 mask);
int8 const_func __attribute__((overloadable)) shuffle2(int16 x, int16 y, uint8 mask);

uint8 const_func __attribute__((overloadable)) shuffle2(uint2 x, uint2 y, uint8 mask);
uint8 const_func __attribute__((overloadable)) shuffle2(uint4 x, uint4 y, uint8 mask);
uint8 const_func __attribute__((overloadable)) shuffle2(uint8 x, uint8 y, uint8 mask);
uint8 const_func __attribute__((overloadable)) shuffle2(uint16 x, uint16 y, uint8 mask);

long8 const_func __attribute__((overloadable)) shuffle2(long2 x, long2 y, ulong8 mask);
long8 const_func __attribute__((overloadable)) shuffle2(long4 x, long4 y, ulong8 mask);
long8 const_func __attribute__((overloadable)) shuffle2(long8 x, long8 y, ulong8 mask);
long8 const_func __attribute__((overloadable)) shuffle2(long16 x, long16 y, ulong8 mask);

ulong8 const_func __attribute__((overloadable)) shuffle2(ulong2 x, ulong2 y, ulong8 mask);
ulong8 const_func __attribute__((overloadable)) shuffle2(ulong4 x, ulong4 y, ulong8 mask);
ulong8 const_func __attribute__((overloadable)) shuffle2(ulong8 x, ulong8 y, ulong8 mask);
ulong8 const_func __attribute__((overloadable)) shuffle2(ulong16 x, ulong16 y, ulong8 mask);

//half8 const_func __attribute__((overloadable)) shuffle2(half2 x, half2 y, ushort8 mask);
//half8 const_func __attribute__((overloadable)) shuffle2(half4 x, half4 y, ushort8 mask);
//half8 const_func __attribute__((overloadable)) shuffle2(half8 x, half8 y, ushort8 mask);
//half8 const_func __attribute__((overloadable)) shuffle2(half16 x, half16 y, ushort8 mask);

float8 const_func __attribute__((overloadable)) shuffle2(float2 x, float2 y, uint8 mask);
float8 const_func __attribute__((overloadable)) shuffle2(float4 x, float4 y, uint8 mask);
float8 const_func __attribute__((overloadable)) shuffle2(float8 x, float8 y, uint8 mask);
float8 const_func __attribute__((overloadable)) shuffle2(float16 x, float16 y, uint8 mask);

double8 const_func __attribute__((overloadable)) shuffle2(double2 x, double2 y, ulong8 mask);
double8 const_func __attribute__((overloadable)) shuffle2(double4 x, double4 y, ulong8 mask);
double8 const_func __attribute__((overloadable)) shuffle2(double8 x, double8 y, ulong8 mask);
double8 const_func __attribute__((overloadable)) shuffle2(double16
```

```
x, double16 y, ulong8 mask);

char16 const_func __attribute__((overloadable)) shuffle2(char2 x, char2 y, uchar16 mask);
char16 const_func __attribute__((overloadable)) shuffle2(char4 x, char4 y, uchar16 mask);
char16 const_func __attribute__((overloadable)) shuffle2(char8 x, char8 y, uchar16 mask);
char16 const_func __attribute__((overloadable)) shuffle2(char16 x, char16 y, uchar16 mask);

uchar16 const_func __attribute__((overloadable)) shuffle2(uchar2 x, uchar2 y, uchar16 mask);
uchar16 const_func __attribute__((overloadable)) shuffle2(uchar4 x, uchar4 y, uchar16 mask);
uchar16 const_func __attribute__((overloadable)) shuffle2(uchar8 x, uchar8 y, uchar16 mask);
uchar16 const_func __attribute__((overloadable)) shuffle2(uchar16 x, uchar16 y, uchar16 mask);

short16 const_func __attribute__((overloadable)) shuffle2(short2 x, short2 y, ushort16 mask);
short16 const_func __attribute__((overloadable)) shuffle2(short4 x, short4 y, ushort16 mask);
short16 const_func __attribute__((overloadable)) shuffle2(short8 x, short8 y, ushort16 mask);
short16 const_func __attribute__((overloadable)) shuffle2(short16 x, short16 y, ushort16 mask);

ushort16 const_func __attribute__((overloadable)) shuffle2(ushort2 x, ushort2 y, ushort16 mask);
ushort16 const_func __attribute__((overloadable)) shuffle2(ushort4 x, ushort4 y, ushort16 mask);
ushort16 const_func __attribute__((overloadable)) shuffle2(ushort8 x, ushort8 y, ushort16 mask);
ushort16 const_func __attribute__((overloadable)) shuffle2(ushort16 x, ushort16 y, ushort16 mask);

int16 const_func __attribute__((overloadable)) shuffle2(int2 x, int2 y, uint16 mask);
int16 const_func __attribute__((overloadable)) shuffle2(int4 x, int4 y, uint16 mask);
int16 const_func __attribute__((overloadable)) shuffle2(int8 x, int8 y, uint16 mask);
int16 const_func __attribute__((overloadable)) shuffle2(int16 x, int16 y, uint16 mask);

uint16 const_func __attribute__((overloadable)) shuffle2(uint2 x, uint2 y, uint16 mask);
uint16 const_func __attribute__((overloadable)) shuffle2(uint4 x, uint4 y, uint16 mask);
uint16 const_func __attribute__((overloadable)) shuffle2(uint8 x, uint8 y, uint16 mask);
uint16 const_func __attribute__((overloadable)) shuffle2(uint16 x, uint16 y, uint16 mask);

long16 const_func __attribute__((overloadable)) shuffle2(long2 x, long2 y, ulong16 mask);
long16 const_func __attribute__((overloadable)) shuffle2(long4 x, long4 y, ulong16 mask);
long16 const_func __attribute__((overloadable)) shuffle2(long8 x, long8 y, ulong16 mask);
long16 const_func __attribute__((overloadable)) shuffle2(long16 x, long16 y, ulong16 mask);

ulong16 const_func __attribute__((overloadable)) shuffle2(ulong2 x, ulong2 y, ulong16 mask);
ulong16 const_func __attribute__((overloadable)) shuffle2(ulong4 x, ulong4 y, ulong16 mask);
ulong16 const_func __attribute__((overloadable)) shuffle2(ulong8 x, ulong8 y, ulong16 mask);
ulong16 const_func __attribute__((overloadable)) shuffle2(ulong16 x, ulong16 y, ulong16 mask);

//half16 const_func __attribute__((overloadable)) shuffle2(half2 x, half2 y, ushort16 mask);
//half16 const_func __attribute__((overloadable)) shuffle2(half4 x, half4 y, ushort16 mask);
//half16 const_func __attribute__((overloadable)) shuffle2(half8 x, half8 y, ushort16 mask);
//half16 const_func __attribute__((overloadable)) shuffle2(half16 x, half16 y, ushort16 mask);

float16 const_func __attribute__((overloadable)) shuffle2(float2 x, float2 y, uint16 mask);
float16 const_func __attribute__((overloadable)) shuffle2(float4 x, float4 y, uint16 mask);
float16 const_func __attribute__((overloadable)) shuffle2(float8 x, float8 y, uint16 mask);
float16 const_func __attribute__((overloadable)) shuffle2(float16 x, float16 y, uint16 mask);
```

```
double16 const_func __attribute__((overloadable)) shuffle2(double2 x, double2 y, ulong16 mask);
double16 const_func __attribute__((overloadable)) shuffle2(double4 x, double4 y, ulong16 mask);
double16 const_func __attribute__((overloadable)) shuffle2(double8 x, double8 y, ulong16 mask);
double16 const_func __attribute__((overloadable)) shuffle2(double16 x, double16 y, ulong16 mask);

// Built-in image functions

/**
 * Use the coordinate (x, y) to do an element lookup in
 * the 2D image object specified by image.
 * read_imagef returns floating-point values in the
 * range [0.0 ... 1.0] for image objects created with
 * image_channel_data_type set to one of the predefined
 * packed formats or CL_UNORM_INT8, or
 * CL_UNORM_INT16.
 * read_imagef returns floating-point values in the
 * range [-1.0 ... 1.0] for image objects created with
 * image_channel_data_type set to CL_SNORM_INT8,
 * or CL_SNORM_INT16.
 * read_imagef returns floating-point values for image
 * objects created with image_channel_data_type set to
 * CL_HALF_FLOAT or CL_FLOAT.
 * The read_imagef calls that take integer coordinates
 * must use a sampler with filter mode set to
 * CLK_FILTER_NEAREST, normalized coordinates set
 * to CLK_NORMALIZED_COORDS_FALSE and
 * addressing mode set to
 * CLK_ADDRESS_CLAMP_TO_EDGE,
 * CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE;
 * otherwise the values returned are undefined.
 * Values returned by read_imagef for image objects
 * with image_channel_data_type values not specified
 * in the description above are undefined.
 */

//
// Addressing Mode.
//
#define CLK_ADDRESS_NONE                0
#define CLK_ADDRESS_CLAMP_TO_EDGE       2
#define CLK_ADDRESS_CLAMP               4
#define CLK_ADDRESS_REPEAT              6
#define CLK_ADDRESS_MIRRORED_REPEAT     8

//
// Coordination Normalization
//
#define CLK_NORMALIZED_COORDS_FALSE     0
#define CLK_NORMALIZED_COORDS_TRUE      1

//
// Filtering Mode.
//
#define CLK_FILTER_NEAREST              0x10
#define CLK_FILTER_LINEAR               0x20

float4 __attribute__((overloadable)) const_func read_imagef(__read_only image2d_t image, sampler_t sampler, int2 coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image2d_t image, sampler_t sampler, float2 coord);
//half4 __attribute__((overloadable)) const_func read_imageh(__read_only image2d_t image, sampler_t sampler, int2 coord);
//half4 __attribute__((overloadable)) const_func read_imageh(__read_only image2d_t image, sampler_t sampler, float2 coord);

//float __attribute__((overloadable)) const_func read_imagef(__read_only image2d_depth_t image, sampler_t sampler, int2 coord);
//float __attribute__((overloadable)) const_func read_imagef(__read_only image2d_depth_t image, sampler_t sampler, float2 coord);
//float __attribute__((overloadable)) const_func read_imagef(__read_only image2d_depth_t image, int2 coord);
//float __attribute__((overloadable)) const_func read_imagef(__read_only image2d_array_depth_t image, sampler_t sampler, int4 coord);
//float __attribute__((overloadable)) const_func read_imagef(__read_only image2d_array_depth_t image, sampler_t sampler, float4 coord);
//float __attribute__((overloadable)) const_func read_imagef(__read_only image2d_array_depth_t image, int4 coord);

//void __attribute__((overloadable)) write_imagef(__write_only image2d_depth_t image, int2 coord, float depth);
//void __attribute__((overloadable)) write_imagef (__write_only image2d_array_depth_t image, int4 coord, float depth);
```

```

//int const_func __attribute__((overloadable)) get_image_width(image2d_depth_t image);
//int const_func __attribute__((overloadable)) get_image_width(image2d_array_depth_t image);

//int const_func __attribute__((overloadable)) get_image_height(image2d_depth_t image);
//int const_func __attribute__((overloadable)) get_image_height(image2d_array_depth_t image);

//int const_func __attribute__((overloadable)) get_image_channel_data_type(image2d_depth_t image);
//int const_func __attribute__((overloadable)) get_image_channel_data_type(image2d_array_depth_t image);

//int const_func __attribute__((overloadable)) get_image_channel_order(image2d_depth_t image);
//int const_func __attribute__((overloadable)) get_image_channel_order(image2d_array_depth_t image);

//int2 const_func __attribute__((overloadable)) get_image_dim(image2d_depth_t image);
//int2 const_func __attribute__((overloadable)) get_image_dim(image2d_array_depth_t image);

//size_t const_func __attribute__((overloadable)) get_image_array_size(image2d_array_depth_t image_array);

/**
 * Use the coordinate (x, y) to do an element lookup in
 * the 2D image object specified by image.
 * read_imagei and read_imageui return
 * unnormalized signed integer and unsigned integer
 * values respectively. Each channel will be stored in a
 * 32-bit integer.
 * read_imagei can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_SIGNED_INT8,
 * CL_SIGNED_INT16 and
 * CL_SIGNED_INT32.
 * If the image_channel_data_type is not one of the
 * above values, the values returned by read_imagei
 * are undefined.
 * read_imageui can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_UNSIGNED_INT8,
 * CL_UNSIGNED_INT16 and
 * CL_UNSIGNED_INT32.
 * If the image_channel_data_type is not one of the
 * above values, the values returned by read_imageui
 * are undefined.
 * The read_image{i|ui} calls support a nearest filter
 * only. The filter_mode specified in sampler
 * must be set to CLK_FILTER_NEAREST; otherwise
 * the values returned are undefined.
 * Furthermore, the read_image{i|ui} calls that take
 * integer coordinates must use a sampler with
 * normalized coordinates set to
 * CLK_NORMALIZED_COORDS_FALSE and
 * addressing mode set to
 * CLK_ADDRESS_CLAMP_TO_EDGE,
 * CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE;
 * otherwise the values returned are undefined.
 */
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image2d_t image, sampler_t sampler, int2 coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image2d_t image, sampler_t sampler, float2 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image2d_t image, sampler_t sampler, int2 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image2d_t image, sampler_t sampler, float2 coord);

/**
 * Write color value to location specified by coordinate
 * (x, y) in the 2D image object specified by image.
 * Appropriate data format conversion to the specified
 * image format is done before writing the color value.
 * x & y are considered to be unnormalized coordinates
 * and must be in the range 0 ... image width - 1, and 0
 * ... image height - 1.
 * write_imagef can only be used with image objects
 * created with image_channel_data_type set to one of
 * the pre-defined packed formats or set to
 * CL_SNORM_INT8, CL_UNORM_INT8,
 * CL_SNORM_INT16, CL_UNORM_INT16,
 * CL_HALF_FLOAT or CL_FLOAT. Appropriate data
 * format conversion will be done to convert channel
 * data from a floating-point value to actual data format
 * in which the channels are stored.
 * write_imagei can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_SIGNED_INT8,
 * CL_SIGNED_INT16 and
```

```
 * CL_SIGNED_INT32.
 * write_imageui can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_UNSIGNED_INT8,
 * CL_UNSIGNED_INT16 and
 * CL_UNSIGNED_INT32.
 * The behavior of write_imagef, write_imagei and
 * write_imageui for image objects created with
 * image_channel_data_type values not specified in
 * the description above or with (x, y) coordinate
 * values that are not in the range (0 ... image width -
 * 1, 0 ... image height - 1), respectively, is undefined.
 */
void __attribute__((overloadable)) write_imagef(__write_only image2d_t image, int2 coord, float4 color);
void __attribute__((overloadable)) write_imagei(__write_only image2d_t image, int2 coord, int4 color);
void __attribute__((overloadable)) write_imageui(__write_only image2d_t image, int2 coord, uint4 color);
void __attribute__((overloadable)) write_imageh(__write_only image2d_t image, int2 coord, half4 color);

/**
 * Use the coordinate (coord.x, coord.y, coord.z) to do
 * an element lookup in the 3D image object specified
 * by image. coord.w is ignored.
 * read_imagef returns floating-point values in the
 * range [0.0 ... 1.0] for image objects created with
 * image_channel_data_type set to one of the predefined
 * packed formats or CL_UNORM_INT8, or
 * CL_UNORM_INT16.
 * read_imagef returns floating-point values in the
 * range [-1.0 ... 1.0] for image objects created with
 * image_channel_data_type set to CL_SNORM_INT8,
 * or CL_SNORM_INT16.
 * read_imagef returns floating-point values for image
 * objects created with image_channel_data_type set to
 * CL_HALF_FLOAT or CL_FLOAT.
 * The read_imagef calls that take integer coordinates
 * must use a sampler with filter mode set to
 * CLK_FILTER_NEAREST, normalized coordinates set
 * to CLK_NORMALIZED_COORDS_FALSE and
 * addressing mode set to
 * CLK_ADDRESS_CLAMP_TO_EDGE,
 * CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE;
 * otherwise the values returned are undefined.
 * Values returned by read_imagef for image objects
 * with image_channel_data_type values not specified
 * in the description are undefined.
 */
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image3d_t image, sampler_t sampler, int4 coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image3d_t image, sampler_t sampler, float4 coord);

/**
 * Use the coordinate (coord.x, coord.y, coord.z) to do
 * an element lookup in the 3D image object specified
 * by image. coord.w is ignored.
 * read_imagei and read_imageui return
 * unnormalized signed integer and unsigned integer
 * values respectively. Each channel will be stored in a
 * 32-bit integer.
 * read_imagei can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_SIGNED_INT8,
 * CL_SIGNED_INT16 and
 * CL_SIGNED_INT32.
 * If the image_channel_data_type is not one of the
 * above values, the values returned by read_imagei
 * are undefined.
 * read_imageui can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_UNSIGNED_INT8,
 * CL_UNSIGNED_INT16 and
 * CL_UNSIGNED_INT32.
 * If the image_channel_data_type is not one of the
 * above values, the values returned by read_imageui
 * are undefined.
 * The read_image{i|ui} calls support a nearest filter
 * only. The filter_mode specified in sampler
 * must be set to CLK_FILTER_NEAREST; otherwise
 * the values returned are undefined.
 * Furthermore, the read_image{i|ui} calls that take
```

```
 * integer coordinates must use a sampler with
 * normalized coordinates set to
 * CLK_NORMALIZED_COORDS_FALSE and
 * addressing mode set to
 * CLK_ADDRESS_CLAMP_TO_EDGE,
 * CLK_ADDRESS_CLAMP or CLK_ADDRESS_NONE;
 * otherwise the values returned are undefined.
 */
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image3d_t image, sampler_t sampler, int4 coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image3d_t image, sampler_t sampler, float4 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image3d_t image, sampler_t sampler, int4 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image3d_t image, sampler_t sampler, float4 coord);

// IMAGE 1.2 built-ins
// with samplers and samplerless
float4  __attribute__((overloadable)) const_func read_imagef(__read_only image2d_array_t image, sampler_t sampler, int4 coord);
float4  __attribute__((overloadable)) const_func read_imagef(__read_only image2d_array_t image, sampler_t sampler, float4 coord);
int4  __attribute__((overloadable)) const_func read_imagei(__read_only image2d_array_t image, sampler_t sampler, int4 coord);
int4  __attribute__((overloadable)) const_func read_imagei(__read_only image2d_array_t image, sampler_t sampler, float4 coord);
uint4  __attribute__((overloadable)) const_func read_imageui(__read_only image2d_array_t image, sampler_t sampler, int4 coord);
uint4  __attribute__((overloadable)) const_func read_imageui(__read_only image2d_array_t image, sampler_t sampler, float4 coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image1d_t image, sampler_t sampler, int coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image1d_t image, sampler_t sampler, float coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_t image, sampler_t sampler, int coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_t image, sampler_t sampler, float coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_t image, sampler_t sampler, int coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_t image, sampler_t sampler, float coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image1d_array_t image, sampler_t sampler, int2 coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image1d_array_t image, sampler_t sampler, float2 coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_array_t image, sampler_t sampler, int2 coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_array_t image, sampler_t sampler, float2 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_array_t image, sampler_t sampler, int2 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_array_t image, sampler_t sampler, float2 coord);
float4 __attribute__((overloadable)) const_func read_imagef (__read_only image2d_t image, int2 coord);
int4 __attribute__((overloadable)) const_func read_imagei (__read_only image2d_t image, int2 coord);
uint4 __attribute__((overloadable)) const_func read_imageui (__read_only image2d_t image, int2 coord);
float4 __attribute__((overloadable)) const_func read_imagef (__read_only image3d_t image, int4 coord);
int4 __attribute__((overloadable)) const_func read_imagei (__read_only image3d_t image, int4 coord);
uint4 __attribute__((overloadable)) const_func read_imageui (__read_only image3d_t image, int4 coord);
float4 __attribute__((overloadable)) const_func read_imagef (__read_only image2d_array_t image, int4 coord);
int4 __attribute__((overloadable)) const_func read_imagei (__read_only image2d_array_t image, int4 coord);
uint4 __attribute__((overloadable)) const_func read_imageui (__read_only image2d_array_t image, int4 coord);
float4 __attribute__((overloadable)) const_func read_imagef (__read_only image1d_t image, int coord);
float4 __attribute__((overloadable)) const_func read_imagef (__read_only image1d_buffer_t image, int coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_t image, int coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_t image, int coord);
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_buffer_t image, int coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_buffer_t image, int coord);
float4 __attribute__((overloadable)) const_func read_imagef(__read_only image1d_array_t image, int2 coord);
```

```
int4 __attribute__((overloadable)) const_func read_imagei(__read_only image1d_array_t image, int2 coord);
uint4 __attribute__((overloadable)) const_func read_imageui(__read_only image1d_array_t image, int2 coord);
void __attribute__((overloadable)) write_imagef (__write_only image2d_array_t image, int4 coord, float4 color);
void __attribute__((overloadable)) write_imagei (__write_only image2d_array_t image, int4 coord, int4 color);
void __attribute__((overloadable)) write_imageui (__write_only image2d_array_t image, int4 coord, uint4 color);
void __attribute__((overloadable)) write_imagef (__write_only image1d_t image, int coord, float4 color);
void __attribute__((overloadable)) write_imagei (__write_only image1d_t image, int coord, int4 color);
void __attribute__((overloadable)) write_imageui (__write_only image1d_t image, int coord, uint4 color);
void __attribute__((overloadable)) write_imagef (__write_only image1d_buffer_t image, int coord, float4 color);
void __attribute__((overloadable)) write_imagei (__write_only image1d_buffer_t image, int coord, int4 color);
void __attribute__((overloadable)) write_imageui (__write_only image1d_buffer_t image, int coord, uint4 color);
void __attribute__((overloadable)) write_imagef (__write_only image1d_array_t image, int2 coord, float4 color);
void __attribute__((overloadable)) write_imagei (__write_only image1d_array_t image, int2 coord, int4 color);
void __attribute__((overloadable)) write_imageui (__write_only image1d_array_t image, int2 coord, uint4 color);

/**
 * Write color value to location specified by coordinate
 * (x, y, z) in the 3D image object specified by image.
 * Appropriate data format conversion to the specified
 * image format is done before writing the color value.
 * x & y are considered to be unnormalized coordinates
 * and must be in the range 0 ... image width - 1, and 0
 * ... image height - 1.
 * write_imagef can only be used with image objects
 * created with image_channel_data_type set to one of
 * the pre-defined packed formats or set to
 * CL_SNORM_INT8, CL_UNORM_INT8,
 * CL_SNORM_INT16, CL_UNORM_INT16,
 * CL_HALF_FLOAT or CL_FLOAT. Appropriate data
 * format conversion will be done to convert channel
 * data from a floating-point value to actual data format
 * in which the channels are stored.
 * write_imagei can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_SIGNED_INT8,
 * CL_SIGNED_INT16 and
 * CL_SIGNED_INT32.
 * write_imageui can only be used with image objects
 * created with image_channel_data_type set to one of
 * the following values:
 * CL_UNSIGNED_INT8,
 * CL_UNSIGNED_INT16 and
 * CL_UNSIGNED_INT32.
 * The behavior of write_imagef, write_imagei and
 * write_imageui for image objects created with
 * image_channel_data_type values not specified in
 * the description above or with (x, y) coordinate
 * values that are not in the range (0 ... image width -
 * 1, 0 ... image height - 1), respectively, is undefined.
 */
void __attribute__((overloadable)) write_imagef(__write_only image3d_t image, int4 coord, float4 color);
void __attribute__((overloadable)) write_imagei(__write_only image3d_t image, int4 coord, int4 color);
void __attribute__((overloadable)) write_imageui(__write_only image3d_t image, int4 coord, uint4 color);
void __attribute__((overloadable)) write_imageh(__write_only image3d_t image, int4 coord, half4 color);

/**
 * Return the image width in pixels.
 */
int const_func __attribute__((overloadable)) get_image_width(image1d_t image);
int const_func __attribute__((overloadable)) get_image_width(image1d_buffer_t image);
int const_func __attribute__((overloadable)) get_image_width(image2d_t image);
int const_func __attribute__((overloadable)) get_image_width(image3d_t image);
int const_func __attribute__((overloadable)) get_image_width(image1d_array_t image);
int const_func __attribute__((overloadable)) get_image_width(image2d_array_t image);

/**
 * Return the image height in pixels.
```

```
 */
int const_func __attribute__((overloadable)) get_image_height(image2d_t image);
int const_func __attribute__((overloadable)) get_image_height(image3d_t image);
int const_func __attribute__((overloadable)) get_image_height(image2d_array_t image);

/**
 * Return the image depth in pixels.
 */
int const_func __attribute__((overloadable)) get_image_depth(image3d_t image);

/**
 * Return the channel data type. Valid values are:
 * CLK_SNORM_INT8
 * CLK_SNORM_INT16
 * CLK_UNORM_INT8
 * CLK_UNORM_INT16
 * CLK_UNORM_SHORT_565
 * CLK_UNORM_SHORT_555
 * CLK_UNORM_SHORT_101010
 * CLK_SIGNED_INT8
 * CLK_SIGNED_INT16
 * CLK_SIGNED_INT32
 * CLK_UNSIGNED_INT8
 * CLK_UNSIGNED_INT16
 * CLK_UNSIGNED_INT32
 * CLK_HALF_FLOAT
 * CLK_FLOAT
 */

// Channel order, numbering must be aligned with cl_channel_order in cl.h
//
// Channel order.
//
#define CLK_R                                       0x10B0
#define CLK_A                                       0x10B1
#define CLK_RG                                      0x10B2
#define CLK_RA                                      0x10B3
#define CLK_RGB                                     0x10B4
#define CLK_RGBA                                    0x10B5
#define CLK_BGRA                                    0x10B6
#define CLK_ARGB                                    0x10B7
#define CLK_INTENSITY                               0x10B8
#define CLK_LUMINANCE                               0x10B9
#define CLK_Rx                                      0x10BA
#define CLK_RGx                                     0x10BB
#define CLK_RGBx                                    0x10BC
#define CLK_DEPTH                                   0x10BD
#define CLK_DEPTH_STENCIL                           0x10BE

//
// Channel Datatype.
//
#define CLK_SNORM_INT8          0x10D0
#define CLK_SNORM_INT16         0x10D1
#define CLK_UNORM_INT8          0x10D2
#define CLK_UNORM_INT16         0x10D3
#define CLK_UNORM_SHORT_565     0x10D4
#define CLK_UNORM_SHORT_555     0x10D5
#define CLK_UNORM_INT_101010    0x10D6
#define CLK_SIGNED_INT8         0x10D7
#define CLK_SIGNED_INT16        0x10D8
#define CLK_SIGNED_INT32        0x10D9
#define CLK_UNSIGNED_INT8       0x10DA
#define CLK_UNSIGNED_INT16      0x10DB
#define CLK_UNSIGNED_INT32      0x10DC
#define CLK_HALF_FLOAT          0x10DD
#define CLK_FLOAT               0x10DE
#define CLK_UNORM_INT24         0x10DF

int const_func __attribute__((overloadable)) get_image_channel_data_type(image1d_t image);
int const_func __attribute__((overloadable)) get_image_channel_data_type(image1d_buffer_t image);
int const_func __attribute__((overloadable)) get_image_channel_data_type(image2d_t image);
int const_func __attribute__((overloadable)) get_image_channel_data_type(image3d_t image);
int const_func __attribute__((overloadable)) get_image_channel_data_type(image1d_array_t image);
int const_func __attribute__((overloadable)) get_image_channel_data_type(image2d_array_t image);

/**
 * Return the image channel order. Valid values are:
 * CLK_A
 * CLK_R
```

```
 * CLK_Rx
 * CLK_RG
 * CLK_RGx
 * CLK_RA
 * CLK_RGB
 * CLK_RGBx
 * CLK_RGBA
 * CLK_ARGB
 * CLK_BGRA
 * CLK_INTENSITY
 * CLK_LUMINANCE
 */
int const_func __attribute__((overloadable)) get_image_channel_order(image1d_t image);
int const_func __attribute__((overloadable)) get_image_channel_order(image1d_buffer_t image);
int const_func __attribute__((overloadable)) get_image_channel_order(image2d_t image);
int const_func __attribute__((overloadable)) get_image_channel_order(image3d_t image);
int const_func __attribute__((overloadable)) get_image_channel_order(image1d_array_t image);
int const_func __attribute__((overloadable)) get_image_channel_order(image2d_array_t image);

/**
 * Return the 2D image width and height as an int2
 * type. The width is returned in the x component, and
 * the height in the y component.
 */
int2 const_func __attribute__((overloadable)) get_image_dim(image2d_t image);
int2 const_func __attribute__((overloadable)) get_image_dim(image2d_array_t image);

/**
 * Return the 3D image width, height, and depth as an
 * int4 type. The width is returned in the x
 * component, height in the y component, depth in the z
 * component and the w component is 0.
 */
int4 const_func __attribute__((overloadable)) get_image_dim(image3d_t image);

/**
 * Return the number of images in the 2D image array.
 */
size_t const_func __attribute__((overloadable)) get_image_array_size(image2d_array_t image_array);

/**
 * Return the number of images in the 1D image array.
 */
size_t const_func __attribute__((overloadable)) get_image_array_size(image1d_array_t image_array);

/**
 *  Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.
 */
float4 const_func __attribute__((overloadable)) read_imagef(__read_only image2d_array_t image_array, sampler_t sampler, int4 coord);

/**
 * Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.
 */
float4 const_func __attribute__((overloadable)) read_imagef(__read_only image2d_array_t image_array, sampler_t sampler, float4 coord);

/**
 * Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.
 */
int4 const_func __attribute__((overloadable)) read_imagei(__read_only image2d_array_t image_array, sampler_t sampler, int4 coord);

/**
 * Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.
 */
int4 const_func __attribute__((overloadable)) read_imagei(__read_only image2d_array_t image_array, sampler_t sampler, float4 coord);

/**
 * Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.
 */
uint4 const_func __attribute__((overloadable)) read_imageui(__read_only image2d_array_t image_array, sampler_t sampler, int4 coord);
```

```
/**
 * Use coord.xy to do an element lookup in the 2D image layer identified by index coord.z in the 2D image array.
 */
uint4 const_func __attribute__((overloadable)) read_imageui(__read_only image2d_array_t image_array, sampler_t sampler, float4 coord);

/**
 * Write color value to location specified by coord.xy in the 2D image layer identified by index coord.z in the 2D image array.
 */
void __attribute__((overloadable)) write_imagef(__write_only image2d_array_t image_array, int4 coord, float4 color);

/**
 * Write color value to location specified by coord.xy in the 2D image layer identified by index coord.z in the 2D image array.
 */
void __attribute__((overloadable)) write_imagei(__write_only image2d_array_t image_array, int4 coord, int4 color);

/**
 * Write color value to location specified by coord.xy in the 2D image layer identified by index coord.z in the 2D image array.
 */
void __attribute__((overloadable)) write_imageui(__write_only image2d_array_t image_array, int4 coord, uint4 color);

/**
 * OpenCL as_typen operators
 * Reinterprets a data type as another data type of the same size
 */

#define as_char(x) __builtin_astype((x), char)
#define as_char2(x) __builtin_astype((x), char2)
#define as_char3(x) __builtin_astype((x), char3)
#define as_char4(x) __builtin_astype((x), char4)
#define as_char8(x) __builtin_astype((x), char8)
#define as_char16(x) __builtin_astype((x), char16)

#define as_uchar(x) __builtin_astype((x), uchar)
#define as_uchar2(x) __builtin_astype((x), uchar2)
#define as_uchar3(x) __builtin_astype((x), uchar3)
#define as_uchar4(x) __builtin_astype((x), uchar4)
#define as_uchar8(x) __builtin_astype((x), uchar8)
#define as_uchar16(x) __builtin_astype((x), uchar16)

#define as_short(x) __builtin_astype((x), short)
#define as_short2(x) __builtin_astype((x), short2)
#define as_short3(x) __builtin_astype((x), short3)
#define as_short4(x) __builtin_astype((x), short4)
#define as_short8(x) __builtin_astype((x), short8)
#define as_short16(x) __builtin_astype((x), short16)

#define as_ushort(x) __builtin_astype((x), ushort)
#define as_ushort2(x) __builtin_astype((x), ushort2)
#define as_ushort3(x) __builtin_astype((x), ushort3)
#define as_ushort4(x) __builtin_astype((x), ushort4)
#define as_ushort8(x) __builtin_astype((x), ushort8)
#define as_ushort16(x) __builtin_astype((x), ushort16)

#define as_int(x) __builtin_astype((x), int)
#define as_int2(x) __builtin_astype((x), int2)
#define as_int3(x) __builtin_astype((x), int3)
#define as_int4(x) __builtin_astype((x), int4)
#define as_int8(x) __builtin_astype((x), int8)
#define as_int16(x) __builtin_astype((x), int16)

#define as_uint(x) __builtin_astype((x), uint)
#define as_uint2(x) __builtin_astype((x), uint2)
#define as_uint3(x) __builtin_astype((x), uint3)
#define as_uint4(x) __builtin_astype((x), uint4)
#define as_uint8(x) __builtin_astype((x), uint8)
#define as_uint16(x) __builtin_astype((x), uint16)

#define as_long(x) __builtin_astype((x), long)
#define as_long2(x) __builtin_astype((x), long2)
#define as_long3(x) __builtin_astype((x), long3)
#define as_long4(x) __builtin_astype((x), long4)
#define as_long8(x) __builtin_astype((x), long8)
#define as_long16(x) __builtin_astype((x), long16)

#define as_ulong(x) __builtin_astype((x), ulong)
#define as_ulong2(x) __builtin_astype((x), ulong2)
#define as_ulong3(x) __builtin_astype((x), ulong3)
#define as_ulong4(x) __builtin_astype((x), ulong4)
```

```
#define as_ulong8(x) __builtin_astype((x), ulong8)
#define as_ulong16(x) __builtin_astype((x), ulong16)

#define as_half(x) __builtin_astype((x), half)
#define as_half2(x) __builtin_astype((x), half2)
#define as_half3(x) __builtin_astype((x), half3)
#define as_half4(x) __builtin_astype((x), half4)
#define as_half8(x) __builtin_astype((x), half8)
#define as_half16(x) __builtin_astype((x), half16)

#define as_float(x) __builtin_astype((x), float)
#define as_float2(x) __builtin_astype((x), float2)
#define as_float3(x) __builtin_astype((x), float3)
#define as_float4(x) __builtin_astype((x), float4)
#define as_float8(x) __builtin_astype((x), float8)
#define as_float16(x) __builtin_astype((x), float16)

#define as_double(x) __builtin_astype((x), double)
#define as_double2(x) __builtin_astype((x), double2)
#define as_double3(x) __builtin_astype((x), double3)
#define as_double4(x) __builtin_astype((x), double4)
#define as_double8(x) __builtin_astype((x), double8)
#define as_double16(x) __builtin_astype((x), double16)

// Explicit conversions

char const_func __attribute__((overloadable)) convert_char_rte(char);
char const_func __attribute__((overloadable)) convert_char_sat_rte(char);
char const_func __attribute__((overloadable)) convert_char_rtz(char);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(char);
char const_func __attribute__((overloadable)) convert_char_rtp(char);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(char);
char const_func __attribute__((overloadable)) convert_char_rtn(char);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(char);
char const_func __attribute__((overloadable)) convert_char(char);
char const_func __attribute__((overloadable)) convert_char_sat(char);
char const_func __attribute__((overloadable)) convert_char_rte(
uchar);
char const_func __attribute__((overloadable)) convert_char_sat_rte(
uchar);
char const_func __attribute__((overloadable)) convert_char_rtz(
uchar);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(
uchar);
char const_func __attribute__((overloadable)) convert_char_rtp(
uchar);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(
uchar);
char const_func __attribute__((overloadable)) convert_char_rtn(
uchar);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(
uchar);
char const_func __attribute__((overloadable)) convert_char(
uchar);
char const_func __attribute__((overloadable)) convert_char_sat(
uchar);
char const_func __attribute__((overloadable)) convert_char_rte(short);
char const_func __attribute__((overloadable)) convert_char_sat_rte(short);
char const_func __attribute__((overloadable)) convert_char_rtz(short);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(short);
char const_func __attribute__((overloadable)) convert_char_rtp(short);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(short);
char const_func __attribute__((overloadable)) convert_char_rtn(short);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(short);
char const_func __attribute__((overloadable)) convert_char(short);
char const_func __attribute__((overloadable)) convert_char_sat(short);
char const_func __attribute__((overloadable)) convert_char_rte(
ushort);
char const_func __attribute__((overloadable)) convert_char_sat_rte(
ushort);
char const_func __attribute__((overloadable)) convert_char_rtz(
ushort);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(
ushort);
char const_func __attribute__((overloadable)) convert_char_rtp(
ushort);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(
ushort);
char const_func __attribute__((overloadable)) convert_char_rtn(
ushort);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(
ushort);
char const_func __attribute__((overloadable)) convert_char(
ushort);
char const_func __attribute__((overloadable)) convert_char_sat(
ushort);
char const_func __attribute__((overloadable)) convert_char_rte(int);
```

```
char const_func __attribute__((overloadable)) convert_char_sat_rte(int);
char const_func __attribute__((overloadable)) convert_char_rtz(int);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(int);
char const_func __attribute__((overloadable)) convert_char_rtp(int);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(int);
char const_func __attribute__((overloadable)) convert_char_rtn(int);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(int);
char const_func __attribute__((overloadable)) convert_char(int);
char const_func __attribute__((overloadable)) convert_char_sat(int);
char const_func __attribute__((overloadable)) convert_char_rte(
uint);
char const_func __attribute__((overloadable)) convert_char_sat_rte(
uint);
char const_func __attribute__((overloadable)) convert_char_rtz(
uint);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(
uint);
char const_func __attribute__((overloadable)) convert_char_rtp(
uint);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(
uint);
char const_func __attribute__((overloadable)) convert_char_rtn(
uint);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(
uint);
char const_func __attribute__((overloadable)) convert_char(
uint);
char const_func __attribute__((overloadable)) convert_char_sat(
uint);
char const_func __attribute__((overloadable)) convert_char_rte(long);
char const_func __attribute__((overloadable)) convert_char_sat_rte(long);
char const_func __attribute__((overloadable)) convert_char_rtz(long);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(long);
char const_func __attribute__((overloadable)) convert_char_rtp(long);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(long);
char const_func __attribute__((overloadable)) convert_char_rtn(long);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(long);
char const_func __attribute__((overloadable)) convert_char(long);
char const_func __attribute__((overloadable)) convert_char_sat(long);
char const_func __attribute__((overloadable)) convert_char_rte(
ulong);
char const_func __attribute__((overloadable)) convert_char_sat_rte(
ulong);
char const_func __attribute__((overloadable)) convert_char_rtz(
ulong);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(
ulong);
char const_func __attribute__((overloadable)) convert_char_rtp(
ulong);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(
ulong);
char const_func __attribute__((overloadable)) convert_char_rtn(
ulong);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(
ulong);
char const_func __attribute__((overloadable)) convert_char(
ulong);
char const_func __attribute__((overloadable)) convert_char_sat(
ulong);
char const_func __attribute__((overloadable)) convert_char_rte(float);
char const_func __attribute__((overloadable)) convert_char_sat_rte(float);
char const_func __attribute__((overloadable)) convert_char_rtz(float);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(float);
char const_func __attribute__((overloadable)) convert_char_rtp(float);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(float);
char const_func __attribute__((overloadable)) convert_char_rtn(float);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(float);
char const_func __attribute__((overloadable)) convert_char(float);
char const_func __attribute__((overloadable)) convert_char_sat(float);
char const_func __attribute__((overloadable)) convert_char_rte(double);
char const_func __attribute__((overloadable)) convert_char_sat_rte(double);
char const_func __attribute__((overloadable)) convert_char_rtz(double);
char const_func __attribute__((overloadable)) convert_char_sat_rtz(double);
char const_func __attribute__((overloadable)) convert_char_rtp(double);
char const_func __attribute__((overloadable)) convert_char_sat_rtp(double);
char const_func __attribute__((overloadable)) convert_char_rtn(double);
char const_func __attribute__((overloadable)) convert_char_sat_rtn(double);
char const_func __attribute__((overloadable)) convert_char(double);
char const_func __attribute__((overloadable)) convert_char_sat(double);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(char);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(char);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(char);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(char);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(char);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(char);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(char);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(char);
```

```
uchar const_func __attribute__((overloadable)) convert_uchar(char);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(char);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(uchar);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(short);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(short);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(short);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(short);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(short);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(short);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(short);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(short);
uchar const_func __attribute__((overloadable)) convert_uchar(short);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(short);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(ushort);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(int);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(int);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(int);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(int);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(int);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(int);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(int);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(int);
uchar const_func __attribute__((overloadable)) convert_uchar(int);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(int);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(uint);
uchar const_func __attribute__((overloadable)) convert_uchar(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(uint);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(long);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(long);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(long);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(long);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(long);
```

```
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(long);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(long);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(long);
uchar const_func __attribute__((overloadable)) convert_uchar(long);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(long);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(
    ulong);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(float);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(float);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(float);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(float);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(float);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(float);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(float);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(float);
uchar const_func __attribute__((overloadable)) convert_uchar(float);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(float);
uchar const_func __attribute__((overloadable)) convert_uchar_rte(double);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rte(double);
uchar const_func __attribute__((overloadable)) convert_uchar_rtz(double);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtz(double);
uchar const_func __attribute__((overloadable)) convert_uchar_rtp(double);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtp(double);
uchar const_func __attribute__((overloadable)) convert_uchar_rtn(double);
uchar const_func __attribute__((overloadable)) convert_uchar_sat_rtn(double);
uchar const_func __attribute__((overloadable)) convert_uchar(double);
uchar const_func __attribute__((overloadable)) convert_uchar_sat(double);

short const_func __attribute__((overloadable)) convert_short_rte(char);
short const_func __attribute__((overloadable)) convert_short_sat_rte(char);
short const_func __attribute__((overloadable)) convert_short_rtz(char);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(char);
short const_func __attribute__((overloadable)) convert_short_rtp(char);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(char);
short const_func __attribute__((overloadable)) convert_short_rtn(char);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(char);
short const_func __attribute__((overloadable)) convert_short(char);
short const_func __attribute__((overloadable)) convert_short_sat(char);
short const_func __attribute__((overloadable)) convert_short_rte(
    uchar);
short const_func __attribute__((overloadable)) convert_short_sat_rte(
    uchar);
short const_func __attribute__((overloadable)) convert_short_rtz(
    uchar);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(
    uchar);
short const_func __attribute__((overloadable)) convert_short_rtp(
    uchar);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(
    uchar);
short const_func __attribute__((overloadable)) convert_short_rtn(
    uchar);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(
    uchar);
short const_func __attribute__((overloadable)) convert_short(
    uchar);
short const_func __attribute__((overloadable)) convert_short_sat(
    uchar);
short const_func __attribute__((overloadable)) convert_short_rte(short);
short const_func __attribute__((overloadable)) convert_short_sat_rte(short);
short const_func __attribute__((overloadable)) convert_short_rtz(short);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(short);
short const_func __attribute__((overloadable)) convert_short_rtp(short);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(short);
short const_func __attribute__((overloadable)) convert_short_rtn(short);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(short);
short const_func __attribute__((overloadable)) convert_short(short);
short const_func __attribute__((overloadable)) convert_short_sat(short);
short const_func __attribute__((overloadable)) convert_short_rte(
```

```
ushort);
short const_func __attribute__((overloadable)) convert_short_sat_rte(ushort);
short const_func __attribute__((overloadable)) convert_short_rtz(ushort);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(ushort);
short const_func __attribute__((overloadable)) convert_short_rtp(ushort);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(ushort);
short const_func __attribute__((overloadable)) convert_short_rtn(ushort);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(ushort);
short const_func __attribute__((overloadable)) convert_short(ushort);
short const_func __attribute__((overloadable)) convert_short_sat(ushort);
short const_func __attribute__((overloadable)) convert_short_rte(int);
short const_func __attribute__((overloadable)) convert_short_sat_rte(int);
short const_func __attribute__((overloadable)) convert_short_rtz(int);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(int);
short const_func __attribute__((overloadable)) convert_short_rtp(int);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(int);
short const_func __attribute__((overloadable)) convert_short_rtn(int);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(int);
short const_func __attribute__((overloadable)) convert_short(int);
short const_func __attribute__((overloadable)) convert_short_sat(int);
short const_func __attribute__((overloadable)) convert_short_rte(uint);
short const_func __attribute__((overloadable)) convert_short_sat_rte(uint);
short const_func __attribute__((overloadable)) convert_short_rtz(uint);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(uint);
short const_func __attribute__((overloadable)) convert_short_rtp(uint);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(uint);
short const_func __attribute__((overloadable)) convert_short_rtn(uint);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(uint);
short const_func __attribute__((overloadable)) convert_short(uint);
short const_func __attribute__((overloadable)) convert_short_sat(uint);
short const_func __attribute__((overloadable)) convert_short_rte(long);
short const_func __attribute__((overloadable)) convert_short_sat_rte(long);
short const_func __attribute__((overloadable)) convert_short_rtz(long);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(long);
short const_func __attribute__((overloadable)) convert_short_rtp(long);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(long);
short const_func __attribute__((overloadable)) convert_short_rtn(long);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(long);
short const_func __attribute__((overloadable)) convert_short(long);
short const_func __attribute__((overloadable)) convert_short_sat(long);
short const_func __attribute__((overloadable)) convert_short_rte(ulong);
short const_func __attribute__((overloadable)) convert_short_sat_rte(ulong);
short const_func __attribute__((overloadable)) convert_short_rtz(ulong);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(ulong);
short const_func __attribute__((overloadable)) convert_short_rtp(ulong);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(ulong);
short const_func __attribute__((overloadable)) convert_short_rtn(ulong);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(ulong);
short const_func __attribute__((overloadable)) convert_short(ulong);
short const_func __attribute__((overloadable)) convert_short_sat(ulong);
short const_func __attribute__((overloadable)) convert_short_rte(float);
short const_func __attribute__((overloadable)) convert_short_sat_rte(float);
short const_func __attribute__((overloadable)) convert_short_rtz(float);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(float);
short const_func __attribute__((overloadable)) convert_short_rtp(float);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(float);
short const_func __attribute__((overloadable)) convert_short_rtn(float);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(float);
```

```
short const_func __attribute__((overloadable)) convert_short(float);
short const_func __attribute__((overloadable)) convert_short_sat(float);
short const_func __attribute__((overloadable)) convert_short_rte(double);
short const_func __attribute__((overloadable)) convert_short_sat_rte(double);
short const_func __attribute__((overloadable)) convert_short_rtz(double);
short const_func __attribute__((overloadable)) convert_short_sat_rtz(double);
short const_func __attribute__((overloadable)) convert_short_rtp(double);
short const_func __attribute__((overloadable)) convert_short_sat_rtp(double);
short const_func __attribute__((overloadable)) convert_short_rtn(double);
short const_func __attribute__((overloadable)) convert_short_sat_rtn(double);
short const_func __attribute__((overloadable)) convert_short(double);
short const_func __attribute__((overloadable)) convert_short_sat(double);

ushort const_func __attribute__((overloadable)) convert_ushort_rte(char);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(char);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(char);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(char);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(char);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(char);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(char);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(char);
ushort const_func __attribute__((overloadable)) convert_ushort(char);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(char);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(
uchar);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(short);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(short);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(short);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(short);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(short);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(short);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(short);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(short);
ushort const_func __attribute__((overloadable)) convert_ushort(short);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(short);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(
ushort);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(int);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(int);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(int);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(int);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(int);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(int);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(int);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(int);
ushort const_func __attribute__((overloadable)) convert_ushort(int);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(int);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(
uint);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(
uint);
```

```
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(uint);
ushort const_func __attribute__((overloadable)) convert_ushort(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(uint);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(long);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(long);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(long);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(long);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(long);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(long);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(long);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(long);
ushort const_func __attribute__((overloadable)) convert_ushort(long);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(long);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(ulong);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(float);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(float);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(float);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(float);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(float);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(float);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(float);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(float);
ushort const_func __attribute__((overloadable)) convert_ushort(float);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(float);
ushort const_func __attribute__((overloadable)) convert_ushort_rte(double);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rte(double);
ushort const_func __attribute__((overloadable)) convert_ushort_rtz(double);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtz(double);
ushort const_func __attribute__((overloadable)) convert_ushort_rtp(double);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtp(double);
ushort const_func __attribute__((overloadable)) convert_ushort_rtn(double);
ushort const_func __attribute__((overloadable)) convert_ushort_sat_rtn(double);
ushort const_func __attribute__((overloadable)) convert_ushort(double);
ushort const_func __attribute__((overloadable)) convert_ushort_sat(double);

int const_func __attribute__((overloadable)) convert_int_rte(char);
int const_func __attribute__((overloadable)) convert_int_sat_rte(char);
int const_func __attribute__((overloadable)) convert_int_rtz(char);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(char);
int const_func __attribute__((overloadable)) convert_int_rtp(char);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(char);
int const_func __attribute__((overloadable)) convert_int_rtn(char);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(char);
int const_func __attribute__((overloadable)) convert_int(char);
int const_func __attribute__((overloadable)) convert_int_sat(char);
int const_func __attribute__((overloadable)) convert_int_rte(uchar);
int const_func __attribute__((overloadable)) convert_int_sat_rte(uchar);
int const_func __attribute__((overloadable)) convert_int_rtz(uchar);
```

```
int const_func __attribute__((overloadable)) convert_int_sat_rtz(uchar);
int const_func __attribute__((overloadable)) convert_int_rtp(uchar);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(uchar);
int const_func __attribute__((overloadable)) convert_int_rtn(uchar);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(uchar);
int const_func __attribute__((overloadable)) convert_int(uchar);
int const_func __attribute__((overloadable)) convert_int_sat(uchar);
int const_func __attribute__((overloadable)) convert_int_rte(short);
int const_func __attribute__((overloadable)) convert_int_sat_rte(short);
int const_func __attribute__((overloadable)) convert_int_rtz(short);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(short);
int const_func __attribute__((overloadable)) convert_int_rtp(short);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(short);
int const_func __attribute__((overloadable)) convert_int_rtn(short);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(short);
int const_func __attribute__((overloadable)) convert_int(short);
int const_func __attribute__((overloadable)) convert_int_sat(short);
int const_func __attribute__((overloadable)) convert_int_rte(ushort);
int const_func __attribute__((overloadable)) convert_int_sat_rte(ushort);
int const_func __attribute__((overloadable)) convert_int_rtz(ushort);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(ushort);
int const_func __attribute__((overloadable)) convert_int_rtp(ushort);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(ushort);
int const_func __attribute__((overloadable)) convert_int_rtn(ushort);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(ushort);
int const_func __attribute__((overloadable)) convert_int(ushort);
int const_func __attribute__((overloadable)) convert_int_sat(ushort);
int const_func __attribute__((overloadable)) convert_int_rte(int);
int const_func __attribute__((overloadable)) convert_int_sat_rte(int);
int const_func __attribute__((overloadable)) convert_int_rtz(int);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(int);
int const_func __attribute__((overloadable)) convert_int_rtp(int);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(int);
int const_func __attribute__((overloadable)) convert_int_rtn(int);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(int);
int const_func __attribute__((overloadable)) convert_int(int);
int const_func __attribute__((overloadable)) convert_int_sat(int);
int const_func __attribute__((overloadable)) convert_int_rte(uint);
int const_func __attribute__((overloadable)) convert_int_sat_rte(uint);
int const_func __attribute__((overloadable)) convert_int_rtz(uint);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(uint);
int const_func __attribute__((overloadable)) convert_int_rtp(uint);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(uint);
int const_func __attribute__((overloadable)) convert_int_rtn(uint);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(uint);
int const_func __attribute__((overloadable)) convert_int(uint);
int const_func __attribute__((overloadable)) convert_int_sat(uint);
int const_func __attribute__((overloadable)) convert_int_rte(long);
int const_func __attribute__((overloadable)) convert_int_sat_rte(long);
int const_func __attribute__((overloadable)) convert_int_rtz(long);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(long);
int const_func __attribute__((overloadable)) convert_int_rtp(long);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(long);
int const_func __attribute__((overloadable)) convert_int_rtn(long);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(long);
int const_func __attribute__((overloadable)) convert_int(long);
int const_func __attribute__((overloadable)) convert_int_sat(long);
int const_func __attribute__((overloadable)) convert_int_rte(ulong);
int const_func __attribute__((overloadable)) convert_int_sat_rte(
```

```
ulong);
int const_func __attribute__((overloadable)) convert_int_rtz(
ulong);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(
ulong);
int const_func __attribute__((overloadable)) convert_int_rtp(
ulong);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(
ulong);
int const_func __attribute__((overloadable)) convert_int_rtn(
ulong);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(
ulong);
int const_func __attribute__((overloadable)) convert_int(
ulong);
int const_func __attribute__((overloadable)) convert_int_sat(
ulong);
int const_func __attribute__((overloadable)) convert_int_rte(float);
int const_func __attribute__((overloadable)) convert_int_sat_rte(float);
int const_func __attribute__((overloadable)) convert_int_rtz(float);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(float);
int const_func __attribute__((overloadable)) convert_int_rtp(float);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(float);
int const_func __attribute__((overloadable)) convert_int_rtn(float);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(float);
int const_func __attribute__((overloadable)) convert_int(float);
int const_func __attribute__((overloadable)) convert_int_sat(float);
int const_func __attribute__((overloadable)) convert_int_rte(double);
int const_func __attribute__((overloadable)) convert_int_sat_rte(double);
int const_func __attribute__((overloadable)) convert_int_rtz(double);
int const_func __attribute__((overloadable)) convert_int_sat_rtz(double);
int const_func __attribute__((overloadable)) convert_int_rtp(double);
int const_func __attribute__((overloadable)) convert_int_sat_rtp(double);
int const_func __attribute__((overloadable)) convert_int_rtn(double);
int const_func __attribute__((overloadable)) convert_int_sat_rtn(double);
int const_func __attribute__((overloadable)) convert_int(double);
int const_func __attribute__((overloadable)) convert_int_sat(double);

uint const_func __attribute__((overloadable)) convert_uint_rte(char);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(char);
uint const_func __attribute__((overloadable)) convert_uint_rtz(char);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(char);
uint const_func __attribute__((overloadable)) convert_uint_rtp(char);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(char);
uint const_func __attribute__((overloadable)) convert_uint_rtn(char);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(char);
uint const_func __attribute__((overloadable)) convert_uint(char);
uint const_func __attribute__((overloadable)) convert_uint_sat(char);
uint const_func __attribute__((overloadable)) convert_uint_rte(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_rtz(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_rtp(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_rtn(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(
uchar);
uint const_func __attribute__((overloadable)) convert_uint(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_sat(
uchar);
uint const_func __attribute__((overloadable)) convert_uint_rte(short);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(short);
uint const_func __attribute__((overloadable)) convert_uint_rtz(short);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(short);
uint const_func __attribute__((overloadable)) convert_uint_rtp(short);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(short);
uint const_func __attribute__((overloadable)) convert_uint_rtn(short);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(short);
uint const_func __attribute__((overloadable)) convert_uint(short);
uint const_func __attribute__((overloadable)) convert_uint_sat(short);
uint const_func __attribute__((overloadable)) convert_uint_rte(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_rtz(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_rtp(
```

```
ushort);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_rtn(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(
ushort);
uint const_func __attribute__((overloadable)) convert_uint(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_sat(
ushort);
uint const_func __attribute__((overloadable)) convert_uint_rte(int);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(int);
uint const_func __attribute__((overloadable)) convert_uint_rtz(int);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(int);
uint const_func __attribute__((overloadable)) convert_uint_rtp(int);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(int);
uint const_func __attribute__((overloadable)) convert_uint_rtn(int);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(int);
uint const_func __attribute__((overloadable)) convert_uint(int);
uint const_func __attribute__((overloadable)) convert_uint_sat(int);
uint const_func __attribute__((overloadable)) convert_uint_rte(
uint);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(
uint);
uint const_func __attribute__((overloadable)) convert_uint_rtz(
uint);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(
uint);
uint const_func __attribute__((overloadable)) convert_uint_rtp(
uint);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(
uint);
uint const_func __attribute__((overloadable)) convert_uint_rtn(
uint);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(
uint);
uint const_func __attribute__((overloadable)) convert_uint(
uint);
uint const_func __attribute__((overloadable)) convert_uint_sat(
uint);
uint const_func __attribute__((overloadable)) convert_uint_rte(long);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(long);
uint const_func __attribute__((overloadable)) convert_uint_rtz(long);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(long);
uint const_func __attribute__((overloadable)) convert_uint_rtp(long);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(long);
uint const_func __attribute__((overloadable)) convert_uint_rtn(long);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(long);
uint const_func __attribute__((overloadable)) convert_uint(long);
uint const_func __attribute__((overloadable)) convert_uint_sat(long);
uint const_func __attribute__((overloadable)) convert_uint_rte(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_rtz(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_rtp(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_rtn(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(
ulong);
uint const_func __attribute__((overloadable)) convert_uint(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_sat(
ulong);
uint const_func __attribute__((overloadable)) convert_uint_rte(float);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(float);
uint const_func __attribute__((overloadable)) convert_uint_rtz(float);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(float);
uint const_func __attribute__((overloadable)) convert_uint_rtp(float);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(float);
uint const_func __attribute__((overloadable)) convert_uint_rtn(float);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(float);
uint const_func __attribute__((overloadable)) convert_uint(float);
uint const_func __attribute__((overloadable)) convert_uint_sat(float);
uint const_func __attribute__((overloadable)) convert_uint_rte(double);
uint const_func __attribute__((overloadable)) convert_uint_sat_rte(double);
uint const_func __attribute__((overloadable)) convert_uint_rtz(double);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtz(double);
uint const_func __attribute__((overloadable)) convert_uint_rtp(double);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtp(double);
```

```
uint const_func __attribute__((overloadable)) convert_uint_rtn(double);
uint const_func __attribute__((overloadable)) convert_uint_sat_rtn(double);
uint const_func __attribute__((overloadable)) convert_uint(double);
uint const_func __attribute__((overloadable)) convert_uint_sat(double);
long const_func __attribute__((overloadable)) convert_long_rte(char);
long const_func __attribute__((overloadable)) convert_long_sat_rte(char);
long const_func __attribute__((overloadable)) convert_long_rtz(char);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(char);
long const_func __attribute__((overloadable)) convert_long_rtp(char);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(char);
long const_func __attribute__((overloadable)) convert_long_rtn(char);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(char);
long const_func __attribute__((overloadable)) convert_long(char);
long const_func __attribute__((overloadable)) convert_long_sat(char);
long const_func __attribute__((overloadable)) convert_long_rte(
uchar);
long const_func __attribute__((overloadable)) convert_long_sat_rte(
uchar);
long const_func __attribute__((overloadable)) convert_long_rtz(
uchar);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(
uchar);
long const_func __attribute__((overloadable)) convert_long_rtp(
uchar);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(
uchar);
long const_func __attribute__((overloadable)) convert_long_rtn(
uchar);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(
uchar);
long const_func __attribute__((overloadable)) convert_long(
uchar);
long const_func __attribute__((overloadable)) convert_long_sat(
uchar);
long const_func __attribute__((overloadable)) convert_long_rte(short);
long const_func __attribute__((overloadable)) convert_long_sat_rte(short);
long const_func __attribute__((overloadable)) convert_long_rtz(short);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(short);
long const_func __attribute__((overloadable)) convert_long_rtp(short);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(short);
long const_func __attribute__((overloadable)) convert_long_rtn(short);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(short);
long const_func __attribute__((overloadable)) convert_long(short);
long const_func __attribute__((overloadable)) convert_long_sat(short);
long const_func __attribute__((overloadable)) convert_long_rte(
ushort);
long const_func __attribute__((overloadable)) convert_long_sat_rte(
ushort);
long const_func __attribute__((overloadable)) convert_long_rtz(
ushort);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(
ushort);
long const_func __attribute__((overloadable)) convert_long_rtp(
ushort);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(
ushort);
long const_func __attribute__((overloadable)) convert_long_rtn(
ushort);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(
ushort);
long const_func __attribute__((overloadable)) convert_long(
ushort);
long const_func __attribute__((overloadable)) convert_long_sat(
ushort);
long const_func __attribute__((overloadable)) convert_long_rte(int);
long const_func __attribute__((overloadable)) convert_long_sat_rte(int);
long const_func __attribute__((overloadable)) convert_long_rtz(int);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(int);
long const_func __attribute__((overloadable)) convert_long_rtp(int);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(int);
long const_func __attribute__((overloadable)) convert_long_rtn(int);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(int);
long const_func __attribute__((overloadable)) convert_long(int);
long const_func __attribute__((overloadable)) convert_long_sat(int);
long const_func __attribute__((overloadable)) convert_long_rte(
uint);
long const_func __attribute__((overloadable)) convert_long_sat_rte(
uint);
long const_func __attribute__((overloadable)) convert_long_rtz(
uint);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(
uint);
long const_func __attribute__((overloadable)) convert_long_rtp(
uint);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(
uint);
long const_func __attribute__((overloadable)) convert_long_rtn(
```

```
uint);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(
uint);
long const_func __attribute__((overloadable)) convert_long(
uint);
long const_func __attribute__((overloadable)) convert_long_sat(
uint);
long const_func __attribute__((overloadable)) convert_long_rte(long);
long const_func __attribute__((overloadable)) convert_long_sat_rte(long);
long const_func __attribute__((overloadable)) convert_long_rtz(long);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(long);
long const_func __attribute__((overloadable)) convert_long_rtp(long);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(long);
long const_func __attribute__((overloadable)) convert_long_rtn(long);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(long);
long const_func __attribute__((overloadable)) convert_long(long);
long const_func __attribute__((overloadable)) convert_long_sat(long);
long const_func __attribute__((overloadable)) convert_long_rte(
ulong);
long const_func __attribute__((overloadable)) convert_long_sat_rte(
ulong);
long const_func __attribute__((overloadable)) convert_long_rtz(
ulong);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(
ulong);
long const_func __attribute__((overloadable)) convert_long_rtp(
ulong);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(
ulong);
long const_func __attribute__((overloadable)) convert_long_rtn(
ulong);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(
ulong);
long const_func __attribute__((overloadable)) convert_long(
ulong);
long const_func __attribute__((overloadable)) convert_long_sat(
ulong);
long const_func __attribute__((overloadable)) convert_long_rte(float);
long const_func __attribute__((overloadable)) convert_long_sat_rte(float);
long const_func __attribute__((overloadable)) convert_long_rtz(float);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(float);
long const_func __attribute__((overloadable)) convert_long_rtp(float);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(float);
long const_func __attribute__((overloadable)) convert_long_rtn(float);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(float);
long const_func __attribute__((overloadable)) convert_long(float);
long const_func __attribute__((overloadable)) convert_long_sat(float);
long const_func __attribute__((overloadable)) convert_long_rte(double);
long const_func __attribute__((overloadable)) convert_long_sat_rte(double);
long const_func __attribute__((overloadable)) convert_long_rtz(double);
long const_func __attribute__((overloadable)) convert_long_sat_rtz(double);
long const_func __attribute__((overloadable)) convert_long_rtp(double);
long const_func __attribute__((overloadable)) convert_long_sat_rtp(double);
long const_func __attribute__((overloadable)) convert_long_rtn(double);
long const_func __attribute__((overloadable)) convert_long_sat_rtn(double);
long const_func __attribute__((overloadable)) convert_long(double);
long const_func __attribute__((overloadable)) convert_long_sat(double);

ulong const_func __attribute__((overloadable)) convert_ulong_rte(char);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(char);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(char);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(char);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(char);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(char);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(char);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(char);
ulong const_func __attribute__((overloadable)) convert_ulong(char);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(char);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong(
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(
```

```
uchar);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(short);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(short);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(short);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(short);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(short);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(short);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(short);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(short);
ulong const_func __attribute__((overloadable)) convert_ulong(short);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(short);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(
ushort);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(int);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(int);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(int);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(int);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(int);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(int);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(int);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(int);
ulong const_func __attribute__((overloadable)) convert_ulong(int);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(int);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(
uint);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(long);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(long);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(long);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(long);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(long);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(long);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(long);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(long);
ulong const_func __attribute__((overloadable)) convert_ulong(long);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(long);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(
ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(
ulong);
```

```
ulong const_func __attribute__((overloadable)) convert_ulong(ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(ulong);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(float);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(float);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(float);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(float);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(float);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(float);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(float);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(float);
ulong const_func __attribute__((overloadable)) convert_ulong(float);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(float);
ulong const_func __attribute__((overloadable)) convert_ulong_rte(double);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rte(double);
ulong const_func __attribute__((overloadable)) convert_ulong_rtz(double);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtz(double);
ulong const_func __attribute__((overloadable)) convert_ulong_rtp(double);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtp(double);
ulong const_func __attribute__((overloadable)) convert_ulong_rtn(double);
ulong const_func __attribute__((overloadable)) convert_ulong_sat_rtn(double);
ulong const_func __attribute__((overloadable)) convert_ulong(double);
ulong const_func __attribute__((overloadable)) convert_ulong_sat(double);

float const_func __attribute__((overloadable)) convert_float_rte(char);
float const_func __attribute__((overloadable)) convert_float_rtz(char);
float const_func __attribute__((overloadable)) convert_float_rtp(char);
float const_func __attribute__((overloadable)) convert_float_rtn(char);
float const_func __attribute__((overloadable)) convert_float(char);
float const_func __attribute__((overloadable)) convert_float_rte(uchar);
float const_func __attribute__((overloadable)) convert_float_rtz(uchar);
float const_func __attribute__((overloadable)) convert_float_rtp(uchar);
float const_func __attribute__((overloadable)) convert_float_rtn(uchar);
float const_func __attribute__((overloadable)) convert_float(uchar);
float const_func __attribute__((overloadable)) convert_float_rte(short);
float const_func __attribute__((overloadable)) convert_float_rtz(short);
float const_func __attribute__((overloadable)) convert_float_rtp(short);
float const_func __attribute__((overloadable)) convert_float_rtn(short);
float const_func __attribute__((overloadable)) convert_float(short);
float const_func __attribute__((overloadable)) convert_float_rte(ushort);
float const_func __attribute__((overloadable)) convert_float_rtz(ushort);
float const_func __attribute__((overloadable)) convert_float_rtp(ushort);
float const_func __attribute__((overloadable)) convert_float_rtn(ushort);
float const_func __attribute__((overloadable)) convert_float(ushort);
float const_func __attribute__((overloadable)) convert_float_rte(int);
float const_func __attribute__((overloadable)) convert_float_rtz(int);
float const_func __attribute__((overloadable)) convert_float_rtp(int);
float const_func __attribute__((overloadable)) convert_float_rtn(int);
float const_func __attribute__((overloadable)) convert_float(int);
float const_func __attribute__((overloadable)) convert_float_rte(uint);
float const_func __attribute__((overloadable)) convert_float_rtz(uint);
float const_func __attribute__((overloadable)) convert_float_rtp(uint);
float const_func __attribute__((overloadable)) convert_float_rtn(uint);
float const_func __attribute__((overloadable)) convert_float(uint);
float const_func __attribute__((overloadable)) convert_float_rte(long);
float const_func __attribute__((overloadable)) convert_float_rtz(long);
float const_func __attribute__((overloadable)) convert_float_rtp(long);
float const_func __attribute__((overloadable)) convert_float_rtn(long);
float const_func __attribute__((overloadable)) convert_float(long);
float const_func __attribute__((overloadable)) convert_float_rte(ulong);
float const_func __attribute__((overloadable)) convert_float_rtz(ulong);
float const_func __attribute__((overloadable)) convert_float_rtp(ulong);
float const_func __attribute__((overloadable)) convert_float_rtn(ulong);
float const_func __attribute__((overloadable)) convert_float(ulong);
float const_func __attribute__((overloadable)) convert_float_rte(float);
float const_func __attribute__((overloadable)) convert_float_rtz(float);
```

```
float const_func __attribute__((overloadable)) convert_float_rtp(float);
float const_func __attribute__((overloadable)) convert_float_rtn(float);
float const_func __attribute__((overloadable)) convert_float(float);
float const_func __attribute__((overloadable)) convert_float_rte(double);
float const_func __attribute__((overloadable)) convert_float_rtz(double);
float const_func __attribute__((overloadable)) convert_float_rtp(double);
float const_func __attribute__((overloadable)) convert_float_rtn(double);
float const_func __attribute__((overloadable)) convert_float(double);

double const_func __attribute__((overloadable)) convert_double_rte(char);
double const_func __attribute__((overloadable)) convert_double_rtz(char);
double const_func __attribute__((overloadable)) convert_double_rtp(char);
double const_func __attribute__((overloadable)) convert_double_rtn(char);
double const_func __attribute__((overloadable)) convert_double(char);
double const_func __attribute__((overloadable)) convert_double_rte(
    uchar);
double const_func __attribute__((overloadable)) convert_double_rtz(
    uchar);
double const_func __attribute__((overloadable)) convert_double_rtp(
    uchar);
double const_func __attribute__((overloadable)) convert_double_rtn(
    uchar);
double const_func __attribute__((overloadable)) convert_double(
    uchar);
double const_func __attribute__((overloadable)) convert_double_rte(short);
double const_func __attribute__((overloadable)) convert_double_rtz(short);
double const_func __attribute__((overloadable)) convert_double_rtp(short);
double const_func __attribute__((overloadable)) convert_double_rtn(short);
double const_func __attribute__((overloadable)) convert_double(short);
double const_func __attribute__((overloadable)) convert_double_rte(
    ushort);
double const_func __attribute__((overloadable)) convert_double_rtz(
    ushort);
double const_func __attribute__((overloadable)) convert_double_rtp(
    ushort);
double const_func __attribute__((overloadable)) convert_double_rtn(
    ushort);
double const_func __attribute__((overloadable)) convert_double(
    ushort);
double const_func __attribute__((overloadable)) convert_double_rte(int);
double const_func __attribute__((overloadable)) convert_double_rtz(int);
double const_func __attribute__((overloadable)) convert_double_rtp(int);
double const_func __attribute__((overloadable)) convert_double_rtn(int);
double const_func __attribute__((overloadable)) convert_double(int);
double const_func __attribute__((overloadable)) convert_double_rte(
    uint);
double const_func __attribute__((overloadable)) convert_double_rtz(
    uint);
double const_func __attribute__((overloadable)) convert_double_rtp(
    uint);
double const_func __attribute__((overloadable)) convert_double_rtn(
    uint);
double const_func __attribute__((overloadable)) convert_double(
    uint);
double const_func __attribute__((overloadable)) convert_double_rte(long);
double const_func __attribute__((overloadable)) convert_double_rtz(long);
double const_func __attribute__((overloadable)) convert_double_rtp(long);
double const_func __attribute__((overloadable)) convert_double_rtn(long);
double const_func __attribute__((overloadable)) convert_double(long);
double const_func __attribute__((overloadable)) convert_double_rte(
    ulong);
double const_func __attribute__((overloadable)) convert_double_rtz(
    ulong);
double const_func __attribute__((overloadable)) convert_double_rtp(
    ulong);
double const_func __attribute__((overloadable)) convert_double_rtn(
    ulong);
double const_func __attribute__((overloadable)) convert_double(
    ulong);
double const_func __attribute__((overloadable)) convert_double_rte(float);
double const_func __attribute__((overloadable)) convert_double_rtz(float);
double const_func __attribute__((overloadable)) convert_double_rtp(float);
double const_func __attribute__((overloadable)) convert_double_rtn(float);
double const_func __attribute__((overloadable)) convert_double(float);
double const_func __attribute__((overloadable)) convert_double_rte(double);
double const_func __attribute__((overloadable)) convert_double_rtz(double);
double const_func __attribute__((overloadable)) convert_double_rtp(double);
double const_func __attribute__((overloadable)) convert_double_rtn(double);
double const_func __attribute__((overloadable)) convert_double(double);

char2 const_func __attribute__((overloadable)) convert_char2_rte(char2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(char2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(char2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(char2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(char2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(char2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(char2);
```

```
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(char2);
char2 const_func __attribute__((overloadable)) convert_char2(char2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(char2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(uchar2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(short2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(short2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(short2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(short2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(short2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(short2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(short2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(short2);
char2 const_func __attribute__((overloadable)) convert_char2(short2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(short2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(ushort2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(int2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(int2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(int2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(int2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(int2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(int2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(int2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(int2);
char2 const_func __attribute__((overloadable)) convert_char2(int2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(int2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(uint2);
char2 const_func __attribute__((overloadable)) convert_char2(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(uint2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(long2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(long2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(long2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(long2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(long2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(long2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(long2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(long2);
char2 const_func __attribute__((overloadable)) convert_char2(long2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(long2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(ulong2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(float2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(float2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(float2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(float2);
char2 const_func __attribute__((overloadable)) convert_char2_rtp(float2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(float2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(float2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(float2);
char2 const_func __attribute__((overloadable)) convert_char2(float2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(float2);
char2 const_func __attribute__((overloadable)) convert_char2_rte(double2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rte(double2);
char2 const_func __attribute__((overloadable)) convert_char2_rtz(double2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtz(double2);
```

```
char2 const_func __attribute__((overloadable)) convert_char2_rtp(double2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtp(double2);
char2 const_func __attribute__((overloadable)) convert_char2_rtn(double2);
char2 const_func __attribute__((overloadable)) convert_char2_sat_rtn(double2);
char2 const_func __attribute__((overloadable)) convert_char2(double2);
char2 const_func __attribute__((overloadable)) convert_char2_sat(double2);

uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(char2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(uchar2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(short2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(ushort2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(int2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(uint2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(long2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(ulong2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(ulong2);
```

```
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(float2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rte(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rte(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtz(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtz(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtp(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtp(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_rtn(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat_rtn(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2(double2);
uchar2 const_func __attribute__((overloadable)) convert_uchar2_sat(double2);

short2 const_func __attribute__((overloadable)) convert_short2_rte(char2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(char2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(char2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(char2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(char2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(char2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(char2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(char2);
short2 const_func __attribute__((overloadable)) convert_short2(char2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(char2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(uchar2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(short2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(short2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(short2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(short2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(short2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(short2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(short2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(short2);
short2 const_func __attribute__((overloadable)) convert_short2(short2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(short2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(ushort2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(int2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(int2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(int2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(int2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(int2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(int2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(int2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(int2);
short2 const_func __attribute__((overloadable)) convert_short2(int2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(int2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(uint2);
short2 const_func __attribute__((overloadable)) convert_short2(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(uint2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(long2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(long2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(long2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(long2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(long2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(long2);
```

```
short2 const_func __attribute__((overloadable)) convert_short2_rtn(long2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(long2);
short2 const_func __attribute__((overloadable)) convert_short2(long2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(long2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(ulong2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(float2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(float2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(float2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(float2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(float2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(float2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(float2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(float2);
short2 const_func __attribute__((overloadable)) convert_short2(float2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(float2);
short2 const_func __attribute__((overloadable)) convert_short2_rte(double2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rte(double2);
short2 const_func __attribute__((overloadable)) convert_short2_rtz(double2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtz(double2);
short2 const_func __attribute__((overloadable)) convert_short2_rtp(double2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtp(double2);
short2 const_func __attribute__((overloadable)) convert_short2_rtn(double2);
short2 const_func __attribute__((overloadable)) convert_short2_sat_rtn(double2);
short2 const_func __attribute__((overloadable)) convert_short2(double2);
short2 const_func __attribute__((overloadable)) convert_short2_sat(double2);

ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(char2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(uchar2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(short2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(ushort2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(int2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(uint2);
```

```
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(uint2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(long2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(ulong2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(float2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rte(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rte(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtz(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtz(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtp(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtp(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_rtn(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat_rtn(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2(double2);
ushort2 const_func __attribute__((overloadable)) convert_ushort2_sat(double2);

int2 const_func __attribute__((overloadable)) convert_int2_rte(char2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(char2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(char2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(char2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(char2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(char2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(char2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(char2);
int2 const_func __attribute__((overloadable)) convert_int2(char2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(char2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(uchar2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(short2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(short2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(short2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(short2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(short2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(short2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(short2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(short2);
int2 const_func __attribute__((overloadable)) convert_int2(short2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(short2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(ushort2);
```

```
int2 const_func __attribute__((overloadable)) convert_int2(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(ushort2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(int2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(int2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(int2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(int2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(int2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(int2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(int2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(int2);
int2 const_func __attribute__((overloadable)) convert_int2(int2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(int2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(uint2);
int2 const_func __attribute__((overloadable)) convert_int2(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(uint2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(long2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(long2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(long2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(long2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(long2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(long2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(long2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(long2);
int2 const_func __attribute__((overloadable)) convert_int2(long2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(long2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(ulong2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(float2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(float2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(float2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(float2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(float2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(float2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(float2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(float2);
int2 const_func __attribute__((overloadable)) convert_int2(float2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(float2);
int2 const_func __attribute__((overloadable)) convert_int2_rte(double2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rte(double2);
int2 const_func __attribute__((overloadable)) convert_int2_rtz(double2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtz(double2);
int2 const_func __attribute__((overloadable)) convert_int2_rtp(double2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtp(double2);
int2 const_func __attribute__((overloadable)) convert_int2_rtn(double2);
int2 const_func __attribute__((overloadable)) convert_int2_sat_rtn(double2);
int2 const_func __attribute__((overloadable)) convert_int2(double2);
int2 const_func __attribute__((overloadable)) convert_int2_sat(double2);

uint2 const_func __attribute__((overloadable)) convert_uint2_rte(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(char2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(uchar2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(short2);
```

```
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(short2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(ushort2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(int2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(uint2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(long2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(ulong2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(float2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rte(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rte(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtz(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtz(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtp(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtp(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_rtn(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat_rtn(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2(double2);
uint2 const_func __attribute__((overloadable)) convert_uint2_sat(double2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(char2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(char2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(char2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(char2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(char2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(char2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(char2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(char2);
long2 const_func __attribute__((overloadable)) convert_long2(char2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(char2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(uchar2);
```

```
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(uchar2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(short2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(short2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(short2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(short2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(short2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(short2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(short2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(short2);
long2 const_func __attribute__((overloadable)) convert_long2(short2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(short2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(ushort2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(int2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(int2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(int2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(int2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(int2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(int2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(int2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(int2);
long2 const_func __attribute__((overloadable)) convert_long2(int2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(int2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(uint2);
long2 const_func __attribute__((overloadable)) convert_long2(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(uint2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(long2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(long2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(long2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(long2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(long2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(long2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(long2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(long2);
long2 const_func __attribute__((overloadable)) convert_long2(long2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(long2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(ulong2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(float2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(float2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(float2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(float2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(float2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(float2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(float2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(float2);
long2 const_func __attribute__((overloadable)) convert_long2(float2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(float2);
long2 const_func __attribute__((overloadable)) convert_long2_rte(double2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rte(double2);
long2 const_func __attribute__((overloadable)) convert_long2_rtz(double2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtz(double2);
long2 const_func __attribute__((overloadable)) convert_long2_rtp(double2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtp(double2);
long2 const_func __attribute__((overloadable)) convert_long2_rtn(double2);
long2 const_func __attribute__((overloadable)) convert_long2_sat_rtn(double2);
```

```
long2 const_func __attribute__((overloadable)) convert_long2(double2);
long2 const_func __attribute__((overloadable)) convert_long2_sat(double2);

ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(char2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(uchar2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(short2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(ushort2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(int2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(uint2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(long2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(ulong2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(float2);
```

```
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(float2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rte(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rte(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtz(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtz(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtp(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtp(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_rtn(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat_rtn(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2(double2);
ulong2 const_func __attribute__((overloadable)) convert_ulong2_sat(double2);

float2 const_func __attribute__((overloadable)) convert_float2_rte(char2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(char2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(char2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(char2);
float2 const_func __attribute__((overloadable)) convert_float2(char2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(uchar2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(uchar2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(uchar2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(uchar2);
float2 const_func __attribute__((overloadable)) convert_float2(uchar2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(short2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(short2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(short2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(short2);
float2 const_func __attribute__((overloadable)) convert_float2(short2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(ushort2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(ushort2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(ushort2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(ushort2);
float2 const_func __attribute__((overloadable)) convert_float2(ushort2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(int2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(int2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(int2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(int2);
float2 const_func __attribute__((overloadable)) convert_float2(int2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(uint2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(uint2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(uint2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(uint2);
float2 const_func __attribute__((overloadable)) convert_float2(uint2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(long2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(long2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(long2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(long2);
float2 const_func __attribute__((overloadable)) convert_float2(long2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(ulong2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(ulong2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(ulong2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(ulong2);
float2 const_func __attribute__((overloadable)) convert_float2(ulong2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(float2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(float2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(float2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(float2);
float2 const_func __attribute__((overloadable)) convert_float2(float2);
float2 const_func __attribute__((overloadable)) convert_float2_rte(double2);
float2 const_func __attribute__((overloadable)) convert_float2_rtz(double2);
float2 const_func __attribute__((overloadable)) convert_float2_rtp(double2);
float2 const_func __attribute__((overloadable)) convert_float2_rtn(double2);
float2 const_func __attribute__((overloadable)) convert_float2(double2);

double2 const_func __attribute__((overloadable)) convert_double2_rte(char2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(char2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(char2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(char2);
double2 const_func __attribute__((overloadable)) convert_double2(char2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(uchar2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(uchar2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(uchar2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(uchar2);
double2 const_func __attribute__((overloadable)) convert_double2(uchar2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(short2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(short2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(short2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(short2);
double2 const_func __attribute__((overloadable)) convert_double2(short2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(ushort2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(ushort2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(ushort2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(ushort2);
```

```
double2 const_func __attribute__((overloadable)) convert_double2(ushort2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(int2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(int2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(int2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(int2);
double2 const_func __attribute__((overloadable)) convert_double2(int2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(uint2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(uint2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(uint2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(uint2);
double2 const_func __attribute__((overloadable)) convert_double2(uint2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(long2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(long2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(long2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(long2);
double2 const_func __attribute__((overloadable)) convert_double2(long2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(ulong2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(ulong2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(ulong2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(ulong2);
double2 const_func __attribute__((overloadable)) convert_double2(ulong2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(float2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(float2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(float2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(float2);
double2 const_func __attribute__((overloadable)) convert_double2(float2);
double2 const_func __attribute__((overloadable)) convert_double2_rte(double2);
double2 const_func __attribute__((overloadable)) convert_double2_rtz(double2);
double2 const_func __attribute__((overloadable)) convert_double2_rtp(double2);
double2 const_func __attribute__((overloadable)) convert_double2_rtn(double2);
double2 const_func __attribute__((overloadable)) convert_double2(double2);

char3 const_func __attribute__((overloadable)) convert_char3_rte(char3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(char3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(char3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(char3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(char3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(char3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(char3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(char3);
char3 const_func __attribute__((overloadable)) convert_char3(char3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(char3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(uchar3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(short3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(short3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(short3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(short3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(short3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(short3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(short3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(short3);
char3 const_func __attribute__((overloadable)) convert_char3(short3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(short3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(ushort3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(int3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(int3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(int3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(int3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(int3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(int3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(int3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(int3);
char3 const_func __attribute__((overloadable)) convert_char3(int3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(int3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(uint3);
```

```
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(uint3);
char3 const_func __attribute__((overloadable)) convert_char3(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(uint3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(long3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(long3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(long3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(long3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(long3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(long3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(long3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(long3);
char3 const_func __attribute__((overloadable)) convert_char3(long3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(long3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(ulong3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(float3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(float3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(float3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(float3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(float3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(float3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(float3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(float3);
char3 const_func __attribute__((overloadable)) convert_char3(float3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(float3);
char3 const_func __attribute__((overloadable)) convert_char3_rte(double3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rte(double3);
char3 const_func __attribute__((overloadable)) convert_char3_rtz(double3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtz(double3);
char3 const_func __attribute__((overloadable)) convert_char3_rtp(double3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtp(double3);
char3 const_func __attribute__((overloadable)) convert_char3_rtn(double3);
char3 const_func __attribute__((overloadable)) convert_char3_sat_rtn(double3);
char3 const_func __attribute__((overloadable)) convert_char3(double3);
char3 const_func __attribute__((overloadable)) convert_char3_sat(double3);

uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(char3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(uchar3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(short3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(ushort3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(int3);
```

```
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(int3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(uint3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(long3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(ulong3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(float3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rte(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rte(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtz(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtz(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtp(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtp(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_rtn(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat_rtn(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3(double3);
uchar3 const_func __attribute__((overloadable)) convert_uchar3_sat(double3);

short3 const_func __attribute__((overloadable)) convert_short3_rte(char3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(char3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(char3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(char3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(char3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(char3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(char3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(char3);
short3 const_func __attribute__((overloadable)) convert_short3(char3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(char3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(uchar3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(short3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(short3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(short3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(short3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(short3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(short3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(short3);
```

```
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(short3);
short3 const_func __attribute__((overloadable)) convert_short3(short3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(short3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(ushort3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(int3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(int3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(int3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(int3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(int3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(int3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(int3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(int3);
short3 const_func __attribute__((overloadable)) convert_short3(int3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(int3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(uint3);
short3 const_func __attribute__((overloadable)) convert_short3(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(uint3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(long3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(long3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(long3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(long3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(long3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(long3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(long3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(long3);
short3 const_func __attribute__((overloadable)) convert_short3(long3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(long3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(ulong3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(float3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(float3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(float3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(float3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(float3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(float3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(float3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(float3);
short3 const_func __attribute__((overloadable)) convert_short3(float3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(float3);
short3 const_func __attribute__((overloadable)) convert_short3_rte(double3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rte(double3);
short3 const_func __attribute__((overloadable)) convert_short3_rtz(double3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtz(double3);
short3 const_func __attribute__((overloadable)) convert_short3_rtp(double3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtp(double3);
short3 const_func __attribute__((overloadable)) convert_short3_rtn(double3);
short3 const_func __attribute__((overloadable)) convert_short3_sat_rtn(double3);
short3 const_func __attribute__((overloadable)) convert_short3(double3);
short3 const_func __attribute__((overloadable)) convert_short3_sat(double3);

ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(char3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(uchar3);
```

```
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(uchar3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(short3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(ushort3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(int3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(uint3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(long3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(ulong3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(float3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rte(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rte(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtz(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtz(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtp(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtp(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_rtn(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat_rtn(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3(double3);
ushort3 const_func __attribute__((overloadable)) convert_ushort3_sat(double3);
```

```

int3 const_func __attribute__((overloadable)) convert_int3_rte(char3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(char3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(char3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(char3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(char3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(char3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(char3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(char3);
int3 const_func __attribute__((overloadable)) convert_int3(char3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(char3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(uchar3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(short3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(short3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(short3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(short3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(short3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(short3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(short3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(short3);
int3 const_func __attribute__((overloadable)) convert_int3(short3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(short3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(ushort3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(int3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(int3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(int3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(int3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(int3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(int3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(int3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(int3);
int3 const_func __attribute__((overloadable)) convert_int3(int3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(int3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(uint3);
int3 const_func __attribute__((overloadable)) convert_int3(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(uint3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(long3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(long3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(long3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(long3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(long3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(long3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(long3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(long3);
int3 const_func __attribute__((overloadable)) convert_int3(long3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(long3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(ulong3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(float3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(float3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(float3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(float3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(float3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(float3);
```

```
int3 const_func __attribute__((overloadable)) convert_int3_rtn(float3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(float3);
int3 const_func __attribute__((overloadable)) convert_int3(float3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(float3);
int3 const_func __attribute__((overloadable)) convert_int3_rte(double3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rte(double3);
int3 const_func __attribute__((overloadable)) convert_int3_rtz(double3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtz(double3);
int3 const_func __attribute__((overloadable)) convert_int3_rtp(double3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtp(double3);
int3 const_func __attribute__((overloadable)) convert_int3_rtn(double3);
int3 const_func __attribute__((overloadable)) convert_int3_sat_rtn(double3);
int3 const_func __attribute__((overloadable)) convert_int3(double3);
int3 const_func __attribute__((overloadable)) convert_int3_sat(double3);

uint3 const_func __attribute__((overloadable)) convert_uint3_rte(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(char3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(uchar3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(short3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(ushort3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(int3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(uint3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(long3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(ulong3);
```

```
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(ulong3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(float3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rte(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rte(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtz(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtz(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtp(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtp(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_rtn(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat_rtn(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3(double3);
uint3 const_func __attribute__((overloadable)) convert_uint3_sat(double3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(char3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(char3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(char3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(char3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(char3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(char3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(char3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(char3);
long3 const_func __attribute__((overloadable)) convert_long3(char3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(char3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(uchar3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(short3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(short3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(short3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(short3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(short3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(short3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(short3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(short3);
long3 const_func __attribute__((overloadable)) convert_long3(short3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(short3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(ushort3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(int3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(int3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(int3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(int3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(int3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(int3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(int3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(int3);
long3 const_func __attribute__((overloadable)) convert_long3(int3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(int3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(uint3);
long3 const_func __attribute__((overloadable)) convert_long3(uint3);
```

```
long3 const_func __attribute__((overloadable)) convert_long3_sat(uint3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(long3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(long3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(long3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(long3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(long3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(long3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(long3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(long3);
long3 const_func __attribute__((overloadable)) convert_long3(long3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(long3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(ulong3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(float3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(float3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(float3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(float3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(float3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(float3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(float3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(float3);
long3 const_func __attribute__((overloadable)) convert_long3(float3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(float3);
long3 const_func __attribute__((overloadable)) convert_long3_rte(double3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rte(double3);
long3 const_func __attribute__((overloadable)) convert_long3_rtz(double3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtz(double3);
long3 const_func __attribute__((overloadable)) convert_long3_rtp(double3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtp(double3);
long3 const_func __attribute__((overloadable)) convert_long3_rtn(double3);
long3 const_func __attribute__((overloadable)) convert_long3_sat_rtn(double3);
long3 const_func __attribute__((overloadable)) convert_long3(double3);
long3 const_func __attribute__((overloadable)) convert_long3_sat(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(char3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(uchar3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(short3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(ushort3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(int3);
```

```
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(int3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(uint3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(long3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(ulong3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(float3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rte(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rte(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtz(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtz(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtp(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtp(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_rtn(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat_rtn(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3(double3);
ulong3 const_func __attribute__((overloadable)) convert_ulong3_sat(double3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(char3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(char3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(char3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(char3);
float3 const_func __attribute__((overloadable)) convert_float3(char3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(uchar3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(uchar3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(uchar3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(uchar3);
float3 const_func __attribute__((overloadable)) convert_float3(uchar3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(short3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(short3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(short3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(short3);
float3 const_func __attribute__((overloadable)) convert_float3(short3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(ushort3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(ushort3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(ushort3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(ushort3);
float3 const_func __attribute__((overloadable)) convert_float3(ushort3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(int3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(int3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(int3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(int3);
float3 const_func __attribute__((overloadable)) convert_float3(int3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(uint3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(uint3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(uint3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(uint3);
float3 const_func __attribute__((overloadable)) convert_float3(uint3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(long3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(long3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(long3);
```

```
float3 const_func __attribute__((overloadable)) convert_float3_rtn(long3);
float3 const_func __attribute__((overloadable)) convert_float3(long3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(ulong3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(ulong3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(ulong3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(ulong3);
float3 const_func __attribute__((overloadable)) convert_float3(ulong3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(float3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(float3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(float3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(float3);
float3 const_func __attribute__((overloadable)) convert_float3(float3);
float3 const_func __attribute__((overloadable)) convert_float3_rte(double3);
float3 const_func __attribute__((overloadable)) convert_float3_rtz(double3);
float3 const_func __attribute__((overloadable)) convert_float3_rtp(double3);
float3 const_func __attribute__((overloadable)) convert_float3_rtn(double3);
float3 const_func __attribute__((overloadable)) convert_float3(double3);

double3 const_func __attribute__((overloadable)) convert_double3_rte(char3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(char3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(char3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(char3);
double3 const_func __attribute__((overloadable)) convert_double3(char3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(uchar3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(uchar3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(uchar3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(uchar3);
double3 const_func __attribute__((overloadable)) convert_double3(uchar3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(short3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(short3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(short3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(short3);
double3 const_func __attribute__((overloadable)) convert_double3(short3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(ushort3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(ushort3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(ushort3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(ushort3);
double3 const_func __attribute__((overloadable)) convert_double3(ushort3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(int3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(int3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(int3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(int3);
double3 const_func __attribute__((overloadable)) convert_double3(int3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(uint3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(uint3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(uint3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(uint3);
double3 const_func __attribute__((overloadable)) convert_double3(uint3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(long3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(long3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(long3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(long3);
double3 const_func __attribute__((overloadable)) convert_double3(long3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(ulong3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(ulong3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(ulong3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(ulong3);
double3 const_func __attribute__((overloadable)) convert_double3(ulong3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(float3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(float3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(float3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(float3);
double3 const_func __attribute__((overloadable)) convert_double3(float3);
double3 const_func __attribute__((overloadable)) convert_double3_rte(double3);
double3 const_func __attribute__((overloadable)) convert_double3_rtz(double3);
double3 const_func __attribute__((overloadable)) convert_double3_rtp(double3);
double3 const_func __attribute__((overloadable)) convert_double3_rtn(double3);
double3 const_func __attribute__((overloadable)) convert_double3(double3);

char4 const_func __attribute__((overloadable)) convert_char4_rte(char4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(char4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(char4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(char4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(char4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(char4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(char4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(char4);
char4 const_func __attribute__((overloadable)) convert_char4(char4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(char4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(uchar4);
```

```
char4 const_func __attribute__((overloadable)) convert_char4(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(uchar4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(short4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(short4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(short4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(short4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(short4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(short4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(short4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(short4);
char4 const_func __attribute__((overloadable)) convert_char4(short4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(short4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(ushort4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(int4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(int4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(int4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(int4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(int4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(int4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(int4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(int4);
char4 const_func __attribute__((overloadable)) convert_char4(int4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(int4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(uint4);
char4 const_func __attribute__((overloadable)) convert_char4(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(uint4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(long4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(long4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(long4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(long4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(long4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(long4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(long4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(long4);
char4 const_func __attribute__((overloadable)) convert_char4(long4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(long4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(ulong4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(float4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(float4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(float4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(float4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(float4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(float4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(float4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(float4);
char4 const_func __attribute__((overloadable)) convert_char4(float4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(float4);
char4 const_func __attribute__((overloadable)) convert_char4_rte(double4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rte(double4);
char4 const_func __attribute__((overloadable)) convert_char4_rtz(double4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtz(double4);
char4 const_func __attribute__((overloadable)) convert_char4_rtp(double4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtp(double4);
char4 const_func __attribute__((overloadable)) convert_char4_rtn(double4);
char4 const_func __attribute__((overloadable)) convert_char4_sat_rtn(double4);
char4 const_func __attribute__((overloadable)) convert_char4(double4);
char4 const_func __attribute__((overloadable)) convert_char4_sat(double4);

uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(char4);
```

```
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(char4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(uchar4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(short4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(ushort4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(int4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(uint4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(long4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(ulong4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(float4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rte(double4);
```

```
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rte(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtz(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtz(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtp(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtp(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_rtn(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat_rtn(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4(double4);
uchar4 const_func __attribute__((overloadable)) convert_uchar4_sat(double4);

short4 const_func __attribute__((overloadable)) convert_short4_rte(char4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(char4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(char4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(char4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(char4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(char4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(char4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(char4);
short4 const_func __attribute__((overloadable)) convert_short4(char4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(char4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(uchar4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(short4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(short4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(short4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(short4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(short4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(short4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(short4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(short4);
short4 const_func __attribute__((overloadable)) convert_short4(short4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(short4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(ushort4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(int4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(int4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(int4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(int4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(int4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(int4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(int4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(int4);
short4 const_func __attribute__((overloadable)) convert_short4(int4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(int4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(uint4);
short4 const_func __attribute__((overloadable)) convert_short4(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(uint4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(long4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(long4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(long4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(long4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(long4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(long4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(long4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(long4);
short4 const_func __attribute__((overloadable)) convert_short4(long4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(long4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(ulong4);
```

```
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(ulong4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(float4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(float4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(float4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(float4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(float4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(float4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(float4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(float4);
short4 const_func __attribute__((overloadable)) convert_short4(float4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(float4);
short4 const_func __attribute__((overloadable)) convert_short4_rte(double4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rte(double4);
short4 const_func __attribute__((overloadable)) convert_short4_rtz(double4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtz(double4);
short4 const_func __attribute__((overloadable)) convert_short4_rtp(double4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtp(double4);
short4 const_func __attribute__((overloadable)) convert_short4_rtn(double4);
short4 const_func __attribute__((overloadable)) convert_short4_sat_rtn(double4);
short4 const_func __attribute__((overloadable)) convert_short4(double4);
short4 const_func __attribute__((overloadable)) convert_short4_sat(double4);

ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(char4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(uchar4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(short4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(ushort4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(int4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(uint4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(long4);
```

```
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(long4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(ulong4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(float4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rte(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rte(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtz(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtz(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtp(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtp(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_rtn(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat_rtn(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4(double4);
ushort4 const_func __attribute__((overloadable)) convert_ushort4_sat(double4);

int4 const_func __attribute__((overloadable)) convert_int4_rte(char4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(char4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(char4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(char4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(char4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(char4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(char4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(char4);
int4 const_func __attribute__((overloadable)) convert_int4(char4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(char4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(uchar4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(short4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(short4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(short4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(short4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(short4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(short4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(short4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(short4);
int4 const_func __attribute__((overloadable)) convert_int4(short4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(short4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(ushort4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(int4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(int4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(int4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(int4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(int4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(int4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(int4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(int4);
int4 const_func __attribute__((overloadable)) convert_int4(int4);
```

```
int4 const_func __attribute__((overloadable)) convert_int4_sat(int4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(uint4);
int4 const_func __attribute__((overloadable)) convert_int4(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(uint4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(long4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(long4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(long4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(long4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(long4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(long4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(long4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(long4);
int4 const_func __attribute__((overloadable)) convert_int4(long4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(long4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(ulong4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(float4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(float4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(float4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(float4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(float4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(float4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(float4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(float4);
int4 const_func __attribute__((overloadable)) convert_int4(float4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(float4);
int4 const_func __attribute__((overloadable)) convert_int4_rte(double4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rte(double4);
int4 const_func __attribute__((overloadable)) convert_int4_rtz(double4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtz(double4);
int4 const_func __attribute__((overloadable)) convert_int4_rtp(double4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtp(double4);
int4 const_func __attribute__((overloadable)) convert_int4_rtn(double4);
int4 const_func __attribute__((overloadable)) convert_int4_sat_rtn(double4);
int4 const_func __attribute__((overloadable)) convert_int4(double4);
int4 const_func __attribute__((overloadable)) convert_int4_sat(double4);

uint4 const_func __attribute__((overloadable)) convert_uint4_rte(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(char4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(uchar4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(short4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(ushort4);
```

```
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(ushort4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(int4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(uint4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(long4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(ulong4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(float4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rte(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rte(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtz(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtz(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtp(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtp(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_rtn(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat_rtn(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4(double4);
uint4 const_func __attribute__((overloadable)) convert_uint4_sat(double4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(char4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(char4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(char4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(char4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(char4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(char4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(char4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(char4);
long4 const_func __attribute__((overloadable)) convert_long4(char4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(char4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(uchar4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(short4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(short4);
```

```
long4 const_func __attribute__((overloadable)) convert_long4_rtz(short4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(short4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(short4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(short4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(short4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(short4);
long4 const_func __attribute__((overloadable)) convert_long4(short4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(short4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(ushort4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(int4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(int4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(int4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(int4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(int4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(int4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(int4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(int4);
long4 const_func __attribute__((overloadable)) convert_long4(int4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(int4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(uint4);
long4 const_func __attribute__((overloadable)) convert_long4(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(uint4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(long4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(long4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(long4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(long4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(long4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(long4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(long4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(long4);
long4 const_func __attribute__((overloadable)) convert_long4(long4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(long4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(ulong4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(float4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(float4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(float4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(float4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(float4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(float4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(float4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(float4);
long4 const_func __attribute__((overloadable)) convert_long4(float4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(float4);
long4 const_func __attribute__((overloadable)) convert_long4_rte(double4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rte(double4);
long4 const_func __attribute__((overloadable)) convert_long4_rtz(double4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtz(double4);
long4 const_func __attribute__((overloadable)) convert_long4_rtp(double4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtp(double4);
long4 const_func __attribute__((overloadable)) convert_long4_rtn(double4);
long4 const_func __attribute__((overloadable)) convert_long4_sat_rtn(double4);
long4 const_func __attribute__((overloadable)) convert_long4(double4);
long4 const_func __attribute__((overloadable)) convert_long4_sat(double4);

ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(char4);
```

```
ulong4 const_func __attribute__((overloadable)) convert_ulong4(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(char4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(uchar4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(short4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(ushort4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(int4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(uint4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(long4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(ulong4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(float4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rte(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rte(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtz(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtz(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtp(double4);
```

```
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtp(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_rtn(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat_rtn(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4(double4);
ulong4 const_func __attribute__((overloadable)) convert_ulong4_sat(double4);

float4 const_func __attribute__((overloadable)) convert_float4_rte(char4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(char4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(char4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(char4);
float4 const_func __attribute__((overloadable)) convert_float4(char4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(uchar4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(uchar4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(uchar4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(uchar4);
float4 const_func __attribute__((overloadable)) convert_float4(uchar4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(short4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(short4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(short4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(short4);
float4 const_func __attribute__((overloadable)) convert_float4(short4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(ushort4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(ushort4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(ushort4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(ushort4);
float4 const_func __attribute__((overloadable)) convert_float4(ushort4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(int4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(int4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(int4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(int4);
float4 const_func __attribute__((overloadable)) convert_float4(int4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(uint4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(uint4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(uint4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(uint4);
float4 const_func __attribute__((overloadable)) convert_float4(uint4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(long4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(long4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(long4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(long4);
float4 const_func __attribute__((overloadable)) convert_float4(long4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(ulong4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(ulong4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(ulong4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(ulong4);
float4 const_func __attribute__((overloadable)) convert_float4(ulong4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(float4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(float4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(float4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(float4);
float4 const_func __attribute__((overloadable)) convert_float4(float4);
float4 const_func __attribute__((overloadable)) convert_float4_rte(double4);
float4 const_func __attribute__((overloadable)) convert_float4_rtz(double4);
float4 const_func __attribute__((overloadable)) convert_float4_rtp(double4);
float4 const_func __attribute__((overloadable)) convert_float4_rtn(double4);
float4 const_func __attribute__((overloadable)) convert_float4(double4);

double4 const_func __attribute__((overloadable)) convert_double4_rte(char4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(char4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(char4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(char4);
double4 const_func __attribute__((overloadable)) convert_double4(char4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(uchar4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(uchar4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(uchar4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(uchar4);
double4 const_func __attribute__((overloadable)) convert_double4(uchar4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(short4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(short4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(short4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(short4);
double4 const_func __attribute__((overloadable)) convert_double4(short4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(ushort4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(ushort4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(ushort4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(ushort4);
double4 const_func __attribute__((overloadable)) convert_double4(ushort4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(int4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(int4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(int4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(int4);
double4 const_func __attribute__((overloadable)) convert_double4(int4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(uint4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(uint4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(uint4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(uint4);
double4 const_func __attribute__((overloadable)) convert_double4(uint4);
```

```
double4 const_func __attribute__((overloadable)) convert_double4_rte(long4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(long4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(long4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(long4);
double4 const_func __attribute__((overloadable)) convert_double4(long4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(ulong4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(ulong4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(ulong4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(ulong4);
double4 const_func __attribute__((overloadable)) convert_double4(ulong4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(float4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(float4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(float4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(float4);
double4 const_func __attribute__((overloadable)) convert_double4(float4);
double4 const_func __attribute__((overloadable)) convert_double4_rte(double4);
double4 const_func __attribute__((overloadable)) convert_double4_rtz(double4);
double4 const_func __attribute__((overloadable)) convert_double4_rtp(double4);
double4 const_func __attribute__((overloadable)) convert_double4_rtn(double4);
double4 const_func __attribute__((overloadable)) convert_double4(double4);

char8 const_func __attribute__((overloadable)) convert_char8_rte(char8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(char8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(char8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(char8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(char8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(char8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(char8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(char8);
char8 const_func __attribute__((overloadable)) convert_char8(char8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(char8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(uchar8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(short8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(short8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(short8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(short8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(short8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(short8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(short8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(short8);
char8 const_func __attribute__((overloadable)) convert_char8(short8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(short8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(ushort8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(int8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(int8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(int8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(int8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(int8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(int8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(int8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(int8);
char8 const_func __attribute__((overloadable)) convert_char8(int8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(int8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(uint8);
char8 const_func __attribute__((overloadable)) convert_char8(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(uint8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(long8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(long8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(long8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(long8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(long8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(long8);
```

```
char8 const_func __attribute__((overloadable)) convert_char8_rtn(long8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(long8);
char8 const_func __attribute__((overloadable)) convert_char8(long8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(long8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(ulong8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(float8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(float8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(float8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(float8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(float8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(float8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(float8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(float8);
char8 const_func __attribute__((overloadable)) convert_char8(float8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(float8);
char8 const_func __attribute__((overloadable)) convert_char8_rte(double8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rte(double8);
char8 const_func __attribute__((overloadable)) convert_char8_rtz(double8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtz(double8);
char8 const_func __attribute__((overloadable)) convert_char8_rtp(double8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtp(double8);
char8 const_func __attribute__((overloadable)) convert_char8_rtn(double8);
char8 const_func __attribute__((overloadable)) convert_char8_sat_rtn(double8);
char8 const_func __attribute__((overloadable)) convert_char8(double8);
char8 const_func __attribute__((overloadable)) convert_char8_sat(double8);

uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(char8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(uchar8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(short8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(ushort8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(int8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(uint8);
```

```
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(uint8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(long8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(ulong8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(float8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rte(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rte(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtz(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtz(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtp(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtp(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_rtn(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat_rtn(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8(double8);
uchar8 const_func __attribute__((overloadable)) convert_uchar8_sat(double8);

short8 const_func __attribute__((overloadable)) convert_short8_rte(char8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(char8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(char8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(char8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(char8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(char8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(char8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(char8);
short8 const_func __attribute__((overloadable)) convert_short8(char8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(char8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(uchar8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(short8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(short8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(short8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(short8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(short8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(short8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(short8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(short8);
short8 const_func __attribute__((overloadable)) convert_short8(short8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(short8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(ushort8);
```

```
short8 const_func __attribute__((overloadable)) convert_short8(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(ushort8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(int8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(int8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(int8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(int8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(int8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(int8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(int8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(int8);
short8 const_func __attribute__((overloadable)) convert_short8(int8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(int8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(uint8);
short8 const_func __attribute__((overloadable)) convert_short8(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(uint8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(long8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(long8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(long8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(long8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(long8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(long8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(long8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(long8);
short8 const_func __attribute__((overloadable)) convert_short8(long8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(long8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(ulong8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(float8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(float8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(float8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(float8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(float8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(float8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(float8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(float8);
short8 const_func __attribute__((overloadable)) convert_short8(float8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(float8);
short8 const_func __attribute__((overloadable)) convert_short8_rte(double8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rte(double8);
short8 const_func __attribute__((overloadable)) convert_short8_rtz(double8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtz(double8);
short8 const_func __attribute__((overloadable)) convert_short8_rtp(double8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtp(double8);
short8 const_func __attribute__((overloadable)) convert_short8_rtn(double8);
short8 const_func __attribute__((overloadable)) convert_short8_sat_rtn(double8);
short8 const_func __attribute__((overloadable)) convert_short8(double8);
short8 const_func __attribute__((overloadable)) convert_short8_sat(double8);

ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(char8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(uchar8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(short8);
```

```
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(short8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(ushort8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(int8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(uint8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(long8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(ulong8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(float8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rte(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rte(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtz(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtz(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtp(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtp(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_rtn(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat_rtn(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8(double8);
ushort8 const_func __attribute__((overloadable)) convert_ushort8_sat(double8);

int8 const_func __attribute__((overloadable)) convert_int8_rte(char8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(char8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(char8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(char8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(char8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(char8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(char8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(char8);
int8 const_func __attribute__((overloadable)) convert_int8(char8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(char8);
```

```
int8 const_func __attribute__((overloadable)) convert_int8_rte(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(uchar8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(short8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(short8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(short8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(short8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(short8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(short8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(short8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(short8);
int8 const_func __attribute__((overloadable)) convert_int8(short8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(short8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(ushort8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(int8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(int8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(int8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(int8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(int8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(int8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(int8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(int8);
int8 const_func __attribute__((overloadable)) convert_int8(int8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(int8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(uint8);
int8 const_func __attribute__((overloadable)) convert_int8(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(uint8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(long8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(long8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(long8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(long8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(long8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(long8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(long8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(long8);
int8 const_func __attribute__((overloadable)) convert_int8(long8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(long8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(ulong8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(float8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(float8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(float8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(float8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(float8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(float8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(float8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(float8);
int8 const_func __attribute__((overloadable)) convert_int8(float8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(float8);
int8 const_func __attribute__((overloadable)) convert_int8_rte(double8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rte(double8);
int8 const_func __attribute__((overloadable)) convert_int8_rtz(double8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtz(double8);
int8 const_func __attribute__((overloadable)) convert_int8_rtp(double8);
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtp(double8);
int8 const_func __attribute__((overloadable)) convert_int8_rtn(double8);
```

```
int8 const_func __attribute__((overloadable)) convert_int8_sat_rtn(double8);
int8 const_func __attribute__((overloadable)) convert_int8(double8);
int8 const_func __attribute__((overloadable)) convert_int8_sat(double8);

uint8 const_func __attribute__((overloadable)) convert_uint8_rte(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(char8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(uchar8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(short8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(ushort8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(int8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(uint8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(long8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(ulong8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(float8);
```

```
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(float8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rte(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rte(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtz(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtz(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtp(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtp(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_rtn(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat_rtn(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8(double8);
uint8 const_func __attribute__((overloadable)) convert_uint8_sat(double8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(char8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(char8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(char8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(char8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(char8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(char8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(char8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(char8);
long8 const_func __attribute__((overloadable)) convert_long8(char8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(char8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(uchar8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(short8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(short8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(short8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(short8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(short8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(short8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(short8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(short8);
long8 const_func __attribute__((overloadable)) convert_long8(short8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(short8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(ushort8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(int8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(int8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(int8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(int8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(int8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(int8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(int8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(int8);
long8 const_func __attribute__((overloadable)) convert_long8(int8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(int8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(uint8);
long8 const_func __attribute__((overloadable)) convert_long8(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(uint8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(long8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(long8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(long8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(long8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(long8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(long8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(long8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(long8);
long8 const_func __attribute__((overloadable)) convert_long8(long8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(long8);
```

```
long8 const_func __attribute__((overloadable)) convert_long8_rte(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(ulong8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(float8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(float8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(float8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(float8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(float8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(float8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(float8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(float8);
long8 const_func __attribute__((overloadable)) convert_long8(float8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(float8);
long8 const_func __attribute__((overloadable)) convert_long8_rte(double8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rte(double8);
long8 const_func __attribute__((overloadable)) convert_long8_rtz(double8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtz(double8);
long8 const_func __attribute__((overloadable)) convert_long8_rtp(double8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtp(double8);
long8 const_func __attribute__((overloadable)) convert_long8_rtn(double8);
long8 const_func __attribute__((overloadable)) convert_long8_sat_rtn(double8);
long8 const_func __attribute__((overloadable)) convert_long8(double8);
long8 const_func __attribute__((overloadable)) convert_long8_sat(double8);

ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(char8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(uchar8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(short8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(ushort8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(int8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(uint8);
```

```
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(uint8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(long8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(ulong8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(float8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rte(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rte(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtz(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtz(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtp(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtp(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_rtn(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat_rtn(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8(double8);
ulong8 const_func __attribute__((overloadable)) convert_ulong8_sat(double8);

float8 const_func __attribute__((overloadable)) convert_float8_rte(char8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(char8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(char8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(char8);
float8 const_func __attribute__((overloadable)) convert_float8(char8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(uchar8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(uchar8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(uchar8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(uchar8);
float8 const_func __attribute__((overloadable)) convert_float8(uchar8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(short8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(short8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(short8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(short8);
float8 const_func __attribute__((overloadable)) convert_float8(short8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(ushort8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(ushort8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(ushort8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(ushort8);
float8 const_func __attribute__((overloadable)) convert_float8(ushort8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(int8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(int8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(int8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(int8);
float8 const_func __attribute__((overloadable)) convert_float8(int8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(uint8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(uint8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(uint8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(uint8);
float8 const_func __attribute__((overloadable)) convert_float8(uint8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(long8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(long8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(long8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(long8);
float8 const_func __attribute__((overloadable)) convert_float8(long8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(ulong8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(ulong8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(ulong8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(ulong8);
float8 const_func __attribute__((overloadable)) convert_float8(ulong8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(float8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(float8);
```

```
float8 const_func __attribute__((overloadable)) convert_float8_rtp(float8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(float8);
float8 const_func __attribute__((overloadable)) convert_float8(float8);
float8 const_func __attribute__((overloadable)) convert_float8_rte(double8);
float8 const_func __attribute__((overloadable)) convert_float8_rtz(double8);
float8 const_func __attribute__((overloadable)) convert_float8_rtp(double8);
float8 const_func __attribute__((overloadable)) convert_float8_rtn(double8);
float8 const_func __attribute__((overloadable)) convert_float8(double8);

double8 const_func __attribute__((overloadable)) convert_double8_rte(char8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(char8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(char8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(char8);
double8 const_func __attribute__((overloadable)) convert_double8(char8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(uchar8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(uchar8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(uchar8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(uchar8);
double8 const_func __attribute__((overloadable)) convert_double8(uchar8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(short8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(short8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(short8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(short8);
double8 const_func __attribute__((overloadable)) convert_double8(short8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(ushort8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(ushort8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(ushort8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(ushort8);
double8 const_func __attribute__((overloadable)) convert_double8(ushort8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(int8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(int8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(int8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(int8);
double8 const_func __attribute__((overloadable)) convert_double8(int8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(uint8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(uint8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(uint8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(uint8);
double8 const_func __attribute__((overloadable)) convert_double8(uint8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(long8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(long8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(long8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(long8);
double8 const_func __attribute__((overloadable)) convert_double8(long8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(ulong8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(ulong8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(ulong8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(ulong8);
double8 const_func __attribute__((overloadable)) convert_double8(ulong8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(float8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(float8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(float8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(float8);
double8 const_func __attribute__((overloadable)) convert_double8(float8);
double8 const_func __attribute__((overloadable)) convert_double8_rte(double8);
double8 const_func __attribute__((overloadable)) convert_double8_rtz(double8);
double8 const_func __attribute__((overloadable)) convert_double8_rtp(double8);
double8 const_func __attribute__((overloadable)) convert_double8_rtn(double8);
double8 const_func __attribute__((overloadable)) convert_double8(double8);
char16 const_func __attribute__((overloadable)) convert_char16_rte(char16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(char16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(char16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(char16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(char16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(char16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(char16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(char16);
char16 const_func __attribute__((overloadable)) convert_char16(char16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(char16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(uchar16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(short16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(short16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(short16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(short16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(short16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(short16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(short16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(short16);
```

```
char16 const_func __attribute__((overloadable)) convert_char16(short16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(short16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(ushort16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(int16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(int16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(int16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(int16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(int16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(int16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(int16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(int16);
char16 const_func __attribute__((overloadable)) convert_char16(int16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(int16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(uint16);
char16 const_func __attribute__((overloadable)) convert_char16(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(uint16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(long16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(long16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(long16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(long16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(long16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(long16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(long16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(long16);
char16 const_func __attribute__((overloadable)) convert_char16(long16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(long16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(ulong16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(float16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(float16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(float16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(float16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(float16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(float16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(float16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(float16);
char16 const_func __attribute__((overloadable)) convert_char16(float16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(float16);
char16 const_func __attribute__((overloadable)) convert_char16_rte(double16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rte(double16);
char16 const_func __attribute__((overloadable)) convert_char16_rtz(double16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtz(double16);
char16 const_func __attribute__((overloadable)) convert_char16_rtp(double16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtp(double16);
char16 const_func __attribute__((overloadable)) convert_char16_rtn(double16);
char16 const_func __attribute__((overloadable)) convert_char16_sat_rtn(double16);
char16 const_func __attribute__((overloadable)) convert_char16(double16);
char16 const_func __attribute__((overloadable)) convert_char16_sat(double16);

uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(char16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(uchar16);
```

```
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(uchar16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(short16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(ushort16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(int16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(uint16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(long16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(ulong16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(float16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rte(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rte(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtz(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtz(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtp(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtp(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_rtn(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat_rtn(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16(double16);
uchar16 const_func __attribute__((overloadable)) convert_uchar16_sat(double16);

```

```
short16 const_func __attribute__((overloadable)) convert_short16_rte(char16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(char16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(char16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(char16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(char16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(char16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(char16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(char16);
short16 const_func __attribute__((overloadable)) convert_short16(char16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(char16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(uchar16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(short16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(short16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(short16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(short16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(short16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(short16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(short16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(short16);
short16 const_func __attribute__((overloadable)) convert_short16(short16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(short16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(ushort16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(int16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(int16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(int16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(int16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(int16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(int16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(int16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(int16);
short16 const_func __attribute__((overloadable)) convert_short16(int16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(int16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(uint16);
short16 const_func __attribute__((overloadable)) convert_short16(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(uint16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(long16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(long16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(long16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(long16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(long16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(long16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(long16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(long16);
short16 const_func __attribute__((overloadable)) convert_short16(long16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(long16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(ulong16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(float16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(float16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(float16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(float16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(float16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(float16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(float16);
```

```
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(float16);
short16 const_func __attribute__((overloadable)) convert_short16(float16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(float16);
short16 const_func __attribute__((overloadable)) convert_short16_rte(double16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rte(double16);
short16 const_func __attribute__((overloadable)) convert_short16_rtz(double16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtz(double16);
short16 const_func __attribute__((overloadable)) convert_short16_rtp(double16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtp(double16);
short16 const_func __attribute__((overloadable)) convert_short16_rtn(double16);
short16 const_func __attribute__((overloadable)) convert_short16_sat_rtn(double16);
short16 const_func __attribute__((overloadable)) convert_short16(double16);
short16 const_func __attribute__((overloadable)) convert_short16_sat(double16);

ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(char16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(uchar16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(short16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(ushort16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(ushort16)
;
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(ushort16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(ushort16)
;
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(ushort16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(ushort16)
;
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(ushort16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(ushort16)
;
ushort16 const_func __attribute__((overloadable)) convert_ushort16(ushort16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(ushort16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(int16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(uint16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(long16);
```

```
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(long16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(ulong16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(float16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rte(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rte(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtz(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtz(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtp(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtp(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_rtn(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat_rtn(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16(double16);
ushort16 const_func __attribute__((overloadable)) convert_ushort16_sat(double16);

int16 const_func __attribute__((overloadable)) convert_int16_rte(char16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(char16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(char16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(char16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(char16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(char16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(char16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(char16);
int16 const_func __attribute__((overloadable)) convert_int16(char16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(char16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(uchar16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(short16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(short16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(short16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(short16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(short16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(short16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(short16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(short16);
int16 const_func __attribute__((overloadable)) convert_int16(short16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(short16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(ushort16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(int16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(int16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(int16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(int16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(int16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(int16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(int16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(int16);
int16 const_func __attribute__((overloadable)) convert_int16(int16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(int16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(uint16);
```

```
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(uint16);
int16 const_func __attribute__((overloadable)) convert_int16(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(uint16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(long16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(long16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(long16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(long16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(long16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(long16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(long16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(long16);
int16 const_func __attribute__((overloadable)) convert_int16(long16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(long16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(ulong16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(float16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(float16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(float16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(float16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(float16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(float16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(float16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(float16);
int16 const_func __attribute__((overloadable)) convert_int16(float16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(float16);
int16 const_func __attribute__((overloadable)) convert_int16_rte(double16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rte(double16);
int16 const_func __attribute__((overloadable)) convert_int16_rtz(double16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtz(double16);
int16 const_func __attribute__((overloadable)) convert_int16_rtp(double16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtp(double16);
int16 const_func __attribute__((overloadable)) convert_int16_rtn(double16);
int16 const_func __attribute__((overloadable)) convert_int16_sat_rtn(double16);
int16 const_func __attribute__((overloadable)) convert_int16(double16);
int16 const_func __attribute__((overloadable)) convert_int16_sat(double16);

uint16 const_func __attribute__((overloadable)) convert_uint16_rte(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(char16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(uchar16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(short16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(ushort16);
```

```
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(ushort16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(int16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(uint16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(long16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(ulong16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(float16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rte(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rte(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtz(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtz(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtp(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtp(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_rtn(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat_rtn(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16(double16);
uint16 const_func __attribute__((overloadable)) convert_uint16_sat(double16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(char16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(char16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(char16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(char16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(char16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(char16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(char16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(char16);
long16 const_func __attribute__((overloadable)) convert_long16(char16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(char16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(uchar16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(short16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(short16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(short16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(short16);
```

```
long16 const_func __attribute__((overloadable)) convert_long16_rtp(short16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(short16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(short16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(short16);
long16 const_func __attribute__((overloadable)) convert_long16(short16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(short16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(ushort16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(int16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(int16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(int16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(int16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(int16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(int16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(int16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(int16);
long16 const_func __attribute__((overloadable)) convert_long16(int16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(int16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(uint16);
long16 const_func __attribute__((overloadable)) convert_long16(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(uint16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(long16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(long16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(long16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(long16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(long16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(long16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(long16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(long16);
long16 const_func __attribute__((overloadable)) convert_long16(long16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(long16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(ulong16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(float16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(float16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(float16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(float16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(float16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(float16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(float16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(float16);
long16 const_func __attribute__((overloadable)) convert_long16(float16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(float16);
long16 const_func __attribute__((overloadable)) convert_long16_rte(double16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rte(double16);
long16 const_func __attribute__((overloadable)) convert_long16_rtz(double16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtz(double16);
long16 const_func __attribute__((overloadable)) convert_long16_rtp(double16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtp(double16);
long16 const_func __attribute__((overloadable)) convert_long16_rtn(double16);
long16 const_func __attribute__((overloadable)) convert_long16_sat_rtn(double16);
long16 const_func __attribute__((overloadable)) convert_long16(double16);
long16 const_func __attribute__((overloadable)) convert_long16_sat(double16);

ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(char16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(char16);
```

```
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(uchar16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(short16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(ushort16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(int16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(uint16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(long16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(ulong16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(float16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rte(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rte(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtz(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtz(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtp(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtp(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_rtn(double16);
```

```
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat_rtn(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16(double16);
ulong16 const_func __attribute__((overloadable)) convert_ulong16_sat(double16);

float16 const_func __attribute__((overloadable)) convert_float16_rte(char16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(char16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(char16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(char16);
float16 const_func __attribute__((overloadable)) convert_float16(char16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(uchar16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(uchar16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(uchar16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(uchar16);
float16 const_func __attribute__((overloadable)) convert_float16(uchar16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(short16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(short16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(short16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(short16);
float16 const_func __attribute__((overloadable)) convert_float16(short16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(ushort16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(ushort16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(ushort16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(ushort16);
float16 const_func __attribute__((overloadable)) convert_float16(ushort16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(int16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(int16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(int16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(int16);
float16 const_func __attribute__((overloadable)) convert_float16(int16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(uint16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(uint16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(uint16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(uint16);
float16 const_func __attribute__((overloadable)) convert_float16(uint16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(long16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(long16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(long16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(long16);
float16 const_func __attribute__((overloadable)) convert_float16(long16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(ulong16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(ulong16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(ulong16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(ulong16);
float16 const_func __attribute__((overloadable)) convert_float16(ulong16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(float16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(float16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(float16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(float16);
float16 const_func __attribute__((overloadable)) convert_float16(float16);
float16 const_func __attribute__((overloadable)) convert_float16_rte(double16);
float16 const_func __attribute__((overloadable)) convert_float16_rtz(double16);
float16 const_func __attribute__((overloadable)) convert_float16_rtp(double16);
float16 const_func __attribute__((overloadable)) convert_float16_rtn(double16);
float16 const_func __attribute__((overloadable)) convert_float16(double16);

double16 const_func __attribute__((overloadable)) convert_double16_rte(char16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(char16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(char16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(char16);
double16 const_func __attribute__((overloadable)) convert_double16(char16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(uchar16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(uchar16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(uchar16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(uchar16);
double16 const_func __attribute__((overloadable)) convert_double16(uchar16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(short16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(short16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(short16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(short16);
double16 const_func __attribute__((overloadable)) convert_double16(short16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(ushort16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(ushort16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(ushort16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(ushort16);
double16 const_func __attribute__((overloadable)) convert_double16(ushort16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(int16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(int16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(int16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(int16);
double16 const_func __attribute__((overloadable)) convert_double16(int16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(uint16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(uint16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(uint16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(uint16);
double16 const_func __attribute__((overloadable)) convert_double16(uint16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(long16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(long16);
```

```
double16 const_func __attribute__((overloadable)) convert_double16_rtp(long16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(long16);
double16 const_func __attribute__((overloadable)) convert_double16(long16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(ulong16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(ulong16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(ulong16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(ulong16);
double16 const_func __attribute__((overloadable)) convert_double16(ulong16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(float16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(float16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(float16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(float16);
double16 const_func __attribute__((overloadable)) convert_double16(float16);
double16 const_func __attribute__((overloadable)) convert_double16_rte(double16);
double16 const_func __attribute__((overloadable)) convert_double16_rtz(double16);
double16 const_func __attribute__((overloadable)) convert_double16_rtp(double16);
double16 const_func __attribute__((overloadable)) convert_double16_rtn(double16);
double16 const_func __attribute__((overloadable)) convert_double16(double16);

#endif
```

11.65 include/CL/sycl/detail/unimplemented.hpp File Reference

```
#include <iostream>
```

Include dependency graph for unimplemented.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- void cl::sycl::detail::unimplemented ()

  Display an "unimplemented" message.

11.66 unimplemented.hpp

```
#ifndef TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
#define TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP

/** \file Deal with unimplemented features
    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <iostream>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup helpers Some helpers for the implementation
```

```
    @{
*/

/** Display an "unimplemented" message

    Can be changed to call assert(0) or whatever.
*/
inline void unimplemented() {
#ifndef NDEBUG
  std::cerr << "Error: using a non implemented feature!!!" << std::endl
            << "Please contribute to the open source implementation. :-)"
            << std::endl;
#endif
}

/// @} End the helpers Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_UNIMPLEMENTED_HPP
```

11.67 include/CL/sycl/device/detail/device.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::device

  An abstract class representing various models of SYCL devices. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.68 device.hpp

```
00001 #ifndef TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
00002 #define TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
00003
00004 /** \file The OpenCL SYCL abstract device
00005
```

```
    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// An abstract class representing various models of SYCL devices
class device {

public:

#ifdef TRISYCL_OPENCL
  /// Return the cl_device_id of the underlying OpenCL platform
  virtual cl_device_id get() const = 0;

  /// Return the underlying Boost.Compute device, if any
  virtual boost::compute::device &get_boost_compute() = 0;
#endif


  /// Return true if the device is a SYCL host device
  virtual bool is_host() const = 0;


  /// Return true if the device is an OpenCL CPU device
  virtual bool is_cpu() const = 0;


  /// Return true if the device is an OpenCL GPU device
  virtual bool is_gpu() const = 0;


  /// Return true if the device is an OpenCL accelerator device
  virtual bool is_accelerator() const = 0;


  /// Return the platform of device
  virtual cl::sycl::platform get_platform() const = 0;


  /// Query the device for OpenCL info::device info
  /** \todo virtual cannot be templated
  template <typename T>
  virtual T get_info(info::device param) const = 0;
  */


  /// Specify whether a specific extension is supported on the device.
  virtual bool has_extension(const string_class &extension) const = 0;


  // Virtual to call the real destructor
  virtual ~device() {}

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_HPP
```

11.69 include/CL/sycl/device.hpp File Reference

```
#include <algorithm>
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#include "CL/sycl/device/detail/opencl_device.hpp"
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::device

SYCL device. More...

- struct std::hash< cl::sycl::device >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

Functions

- template<>
  auto cl::sycl::device::get_info< info::device::max_work_group_size > () const
- template<>
  auto cl::sycl::device::get_info< info::device::max_compute_units > () const
- template<>
  auto cl::sycl::device::get_info< info::device::device_type > () const
- template<>
  auto cl::sycl::device::get_info< info::device::local_mem_size > () const
- template<>
  auto cl::sycl::device::get_info< info::device::max_mem_alloc_size > () const
- template<>
  auto cl::sycl::device::get_info< info::device::vendor > () const
- template<>
  auto cl::sycl::device::get_info< info::device::name > () const
- template<>
  auto cl::sycl::device::get_info< info::device::profile > () const

11.70 device.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_HPP

/** \file The OpenCL SYCL device

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/device/detail/opencl_device.hpp"
#endif
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {

```

```
class device_selector;
class platform;

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// SYCL device
class device
  /* Use the underlying device implementation that can be shared in the
     SYCL model */
  : public detail::shared_ptr_implementation<device, detail::device> {

  // The type encapsulating the implementation
  using implementation_t =
    detail::shared_ptr_implementation<device, detail::device>
;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /// The default constructor uses the SYCL host device
  device() : implementation_t {
detail::host_device::instance() } {}


#ifdef TRISYCL_OPENCL
  /** Construct a device class instance using cl_device_id of the
      OpenCL device

      Return synchronous errors via the SYCL exception class.

      Retain a reference to the OpenCL device and if this device was
      an OpenCL subdevice the device should be released by the caller
      when it is no longer needed.
  */
  device(cl_device_id device_id)
    : device { boost::compute::device { device_id } } {}


  /** Construct a device class instance using a boost::compute::device

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.
  */
  device(const boost::compute::device &d)
    : implementation_t { detail::opencl_device::instance(d)
} {}
#endif


  /** Construct a device class instance using the device selector
      provided

      Return errors via C++ exception class.

      \todo Make it non-explicit in the specification?
  */
  explicit device(const device_selector &ds) {
    auto devices = device::get_devices();
    if (devices.empty())
      // \todo Put a SYCL exception
      throw std::domain_error("No device at all! Internal error...");

    /* Find the device with the best score according to the given
       device_selector */
    auto max = std::max_element(devices.cbegin(), devices.cend(),
                                [&] (const device &d1, const device &d2) {
                                  return ds(d1) < ds(d2);
                                });
    if (ds(*max) < 0)
      // \todo Put a SYCL exception
      throw std::domain_error("No device selected because no positive "
                              "device_selector score found");

    // Create the current device as a shared copy of the selected one
    implementation = max->implementation;
  }


#ifdef TRISYCL_OPENCL
  /** Return the cl_device_id of the underlying OpenCL platform

      Return synchronous errors via the SYCL exception class.
```

```

      Retain a reference to the returned cl_device_id object. Caller
      should release it when finished.

      In the case where this is the SYCL host device it will throw an
      exception.
  */
  cl_device_id get() const {
    return implementation->get();
  }


  /** Return the underlying Boost.Compute device if it is an
      OpenCL device

      This is a triSYCL extension
  */
  boost::compute::device get_boost_compute() const {
    return implementation->get_boost_compute();
  }
#endif


  /// Return true if the device is the SYCL host device
  bool is_host() const {
    return implementation->is_host();
  }


  /// Return true if the device is an OpenCL CPU device
  bool is_cpu() const {
    return implementation->is_cpu();
  }


  /// Return true if the device is an OpenCL GPU device
  bool is_gpu() const {
    return implementation->is_gpu();
  }


  /// Return true if the device is an OpenCL accelerator device
  bool is_accelerator() const {
    return implementation->is_accelerator();
  }



  /** Return the device_type of a device

      \todo Present in Boost.Compute, to be added to the specification
  */
  info::device_type type() const {
    if (is_host())
      return info::device_type::host;
    else if (is_cpu())
      return info::device_type::cpu;
    else if (is_gpu())
      return info::device_type::gpu;
    else if (is_accelerator())
      return info::device_type::accelerator;
    else
      // \todo Put a SYCL exception
      throw std::domain_error("Unknown cl::sycl::info::device_type");
  }


  /** Return the platform of device

      Return synchronous errors via the SYCL exception class.
  */
  platform get_platform() const {
    return implementation->get_platform();
  }


  /** Return a list of all available devices

      Return synchronous errors via SYCL exception classes.
  */
#ifdef _MSC_VER
  inline
#endif
  static vector_class<device>
    get_devices(info::device_type device_type =
info::device_type::all)
    TRISYCL_WEAK_ATTRIB_SUFFIX;
```

```

  /** Query the device for OpenCL info::device info

      Return synchronous errors via the SYCL exception class.

      \todo
  */
  template <typename T>
  T get_info(info::device param) const {
    //return implementation->get_info<Param>(param);
  }


  /** Query the device for OpenCL info::device info

      Return synchronous errors via the SYCL exception class.

      \todo
  */
  template <info::device Param>
  inline auto get_info() const;
  /*{
    // Forward to the version where the info parameter is not a template
    //return get_info<typename info::param_traits_t<info::device, Param>>(Param);
    detail::unimplemented();
    return 0;
  }*/


  /// Test if a specific extension is supported on the device
  bool has_extension(const string_class &extension) const {
    return implementation->has_extension(extension);
  }


#ifdef XYZTRISYCL_OPENCL
  /** Partition the device into sub devices based upon the properties
      provided

      Return synchronous errors via SYCL exception classes.

      \todo
  */
  vector_class<device>
  create_sub_devices(info::device_partition_type partition_type,
                     info::device_partition_property partition_property,
                     info::device_affinity_domain affinity_domain) const {
    return implementation->create_sub_devices(partition_type,
                                              partition_property,
                                              affinity_domain);
  }
#endif

};


template <>
inline auto device::get_info<info::device::max_work_group_size>() const {
  return size_t { 8 };
}

template <>
inline auto device::get_info<info::device::max_compute_units>() const {
  return size_t { 8 };
}

template <>
inline auto device::get_info<info::device::device_type>() const {
  return info::device_type::cpu;
}

template <>
inline auto device::get_info<info::device::local_mem_size>() const {
  return size_t { 32000 };
}

template <>
inline auto device::get_info<info::device::max_mem_alloc_size>() const {
  return size_t { 32000 };
}

template <>
inline auto device::get_info<info::device::vendor>() const {
  return string_class {};
}

template <>
```

```
inline auto device::get_info<info::device::name>() const {
  return string_class {};
}

template <>
inline auto device::get_info<info::device::profile>() const {
  return string_class { "FULL_PROFILE" };
}

/// @} to end the Doxygen group

}
}


/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::device> {

  auto operator()(const cl::sycl::device &d) const {
    // Forward the hashing to the implementation
    return d.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_HPP
```

11.71 include/CL/sycl/info/device.hpp File Reference

`#include "CL/sycl/info/param_traits.hpp"`

Include dependency graph for device.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Typedefs

- using cl::sycl::info::device_fp_config = unsigned int
- using cl::sycl::info::device_exec_capabilities = unsigned int
- using cl::sycl::info::device_queue_properties = unsigned int

Enumerations

- enum cl::sycl::info::device_type : unsigned int { cl::sycl::info::device_type::cpu, cl::sycl::info::device_type::gpu, cl::sycl::info::device_type::accelerator, cl::sycl::info::device_type::custom, cl::sycl::info::device_type::automatic, cl::sycl::info::device_type::host, cl::sycl::info::device_type::opencl, cl::sycl::info::device_type::all }

  Type of devices.

- enum cl::sycl::info::device : int {
  cl::sycl::info::device::device_type, cl::sycl::info::device::vendor_id, cl::sycl::info::device::max_compute_units, cl::sycl::info::device::max_work_item_dimensions, cl::sycl::info::device::max_work_item_sizes, cl::sycl::info::device::max_work_group_size, cl::sycl::info::device::preferred_vector_width_char, cl::sycl::info::device::preferred_vector_width_short, cl::sycl::info::device::preferred_vector_width_int, cl::sycl::info::device::preferred_vector_width_long_long, cl::sycl::info::device::preferred_vector_width_float, cl::sycl::info::device::preferred_vector_width_double, cl::sycl::info::device::preferred_vector_width_half, cl::sycl::info::device::native_vector_witdth_char, cl::sycl::info::device::native_vector_witdth_short, cl::sycl::info::device::native_vector_witdth_int, cl::sycl::info::device::native_vector_witdth_long_long, cl::sycl::info::device::native_vector_witdth_float, cl::sycl::info::device::native_vector_witdth_double, cl::sycl::info::device::native_vector_witdth_half, cl::sycl::info::device::max_clock_frequency, cl::sycl::info::device::address_bits, cl::sycl::info::device::max_mem_alloc_size, cl::sycl::info::device::image_support, cl::sycl::info::device::max_read_image_args, cl::sycl::info::device::max_write_image_args, cl::sycl::info::device::image2d_max_height, cl::sycl::info::device::image2d_max_width, cl::sycl::info::device::image3d_max_height, cl::sycl::info::device::image3d_max_widht, cl::sycl::info::device::image3d_mas_depth, cl::sycl::info::device::image_max_buffer_size, cl::sycl::info::device::image_max_array_size, cl::sycl::info::device::max_samplers, cl::sycl::info::device::max_parameter_size, cl::sycl::info::device::mem_base_addr_align, cl::sycl::info::device::single_fp_config, cl::sycl::info::device::double_fp_config, cl::sycl::info::device::global_mem_cache_type, cl::sycl::info::device::global_mem_cache_line_size, cl::sycl::info::device::global_mem_cache_size, cl::sycl::info::device::global_mem_size, cl::sycl::info::device::max_constant_buffer_size, cl::sycl::info::device::max_constant_args, cl::sycl::info::device::local_mem_type, cl::sycl::info::device::local_mem_size, cl::sycl::info::device::error_correction_support, cl::sycl::info::device::host_unified_memory, cl::sycl::info::device::profiling_timer_resolution, cl::sycl::info::device::endian_little, cl::sycl::info::device::is_available, cl::sycl::info::device::is_compiler_available, cl::sycl::info::device::is_linker_available, cl::sycl::info::device::execution_capabilities, cl::sycl::info::device::queue_properties, cl::sycl::info::device::built_in_kernels, cl::sycl::info::device::platform, cl::sycl::info::device::name, cl::sycl::info::device::vendor, cl::sycl::info::device::driver_version, cl::sycl::info::device::profile, cl::sycl::info::device::device_version, cl::sycl::info::device::opencl_version, cl::sycl::info::device::extensions, cl::sycl::info::device::printf_buffer_size, cl::sycl::info::device::preferred_interop_user_sync, cl::sycl::info::device::parent_device, cl::sycl::info::device::partition_max_sub_devices, cl::sycl::info::device::partition_properties, cl::sycl::info::device::partition_affinity_domain, cl::sycl::info::device::partition_type, cl::sycl::info::device::reference_count }

  Device information descriptors.
- enum cl::sycl::info::device_partition_property : int {
  cl::sycl::info::device_partition_property::unsupported, cl::sycl::info::device_partition_property::partition_equally, cl::sycl::info::device_partition_property::partition_by_counts, cl::sycl::info::device_partition_property::partition_by_affinity_domain, cl::sycl::info::device_partition_property::partition_affinity_domain_next_partitionable }
- enum cl::sycl::info::device_affinity_domain : int {
  cl::sycl::info::device_affinity_domain::unsupported, cl::sycl::info::device_affinity_domain::numa, cl::sycl::info::device_affinity_domain::L4_cache, cl::sycl::info::device_affinity_domain::L3_cache, cl::sycl::info::device_affinity_domain::L2_cache, cl::sycl::info::device_affinity_domain::next_partitionable }
- enum cl::sycl::info::device_partition_type : int {
  cl::sycl::info::device_partition_type::no_partition, cl::sycl::info::device_partition_type::numa, cl::sycl::info::device_partition_type::L4_cache, cl::sycl::info::device_partition_type::L3_cache, cl::sycl::info::device_partition_type::L2_cache, cl::sycl::info::device_partition_type::L1_cache }
- enum cl::sycl::info::local_mem_type : int { cl::sycl::info::local_mem_type::none, cl::sycl::info::local_mem_type::local, cl::sycl::info::local_mem_type::global }
- enum cl::sycl::info::fp_config : int {
  cl::sycl::info::fp_config::denorm, cl::sycl::info::fp_config::inf_nan, cl::sycl::info::fp_config::round_to_nearest, cl::sycl::info::fp_config::round_to_zero, cl::sycl::info::fp_config::round_to_inf, cl::sycl::info::fp_config::fma, cl::sycl::info::fp_config::correctly_rounded_divide_sqrt, cl::sycl::info::fp_config::soft_float }

- enum cl::sycl::info::global_mem_cache_type : int { cl::sycl::info::global_mem_cache_type::none, cl::sycl↩ ::info::global_mem_cache_type::read_only, cl::sycl::info::global_mem_cache_type::write_only }
- enum cl::sycl::info::device_execution_capabilities : unsigned int { cl::sycl::info::device_execution_↩ capabilities::exec_kernel, cl::sycl::info::device_execution_capabilities::exec_native_kernel }

11.72 device.hpp

```
#ifndef TRISYCL_SYCL_INFO_DEVICE_HPP
#define TRISYCL_SYCL_INFO_DEVICE_HPP

/** \file The OpenCL SYCL device information parameters

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/info/param_traits.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

namespace info {

/** Type of devices

    To be used either to define a device type or to select more
    broadly a kind of device

    \todo To be moved in the specification from platform to device

    \todo Add opencl to the specification

    \todo there is no accelerator_selector and custom_accelerator
*/
enum class device_type : unsigned int {
  cpu,
  gpu,
  accelerator,
  custom,
  automatic,
  host,
  opencl,
  all
};


/** Device information descriptors

    From specs/latex/headers/deviceInfo.h in the specification

    \todo Should be unsigned int?
*/
enum class device : int {
  device_type,
  vendor_id,
  max_compute_units,
  max_work_item_dimensions,
  max_work_item_sizes,
  max_work_group_size,
  preferred_vector_width_char,
  preferred_vector_width_short,
  preferred_vector_width_int,
  preferred_vector_width_long_long,
  preferred_vector_width_float,
  preferred_vector_width_double,
  preferred_vector_width_half,
  native_vector_witdth_char,
  native_vector_witdth_short,
  native_vector_witdth_int,
  native_vector_witdth_long_long,
  native_vector_witdth_float,
  native_vector_witdth_double,
  native_vector_witdth_half,
  max_clock_frequency,
```

```
  address_bits,
  max_mem_alloc_size,
  image_support,
  max_read_image_args,
  max_write_image_args,
  image2d_max_height,
  image2d_max_width,
  image3d_max_height,
  image3d_max_widht,
  image3d_mas_depth,
  image_max_buffer_size,
  image_max_array_size,
  max_samplers,
  max_parameter_size,
  mem_base_addr_align,
  single_fp_config,
  double_fp_config,
  global_mem_cache_type,
  global_mem_cache_line_size,
  global_mem_cache_size,
  global_mem_size,
  max_constant_buffer_size,
  max_constant_args,
  local_mem_type,
  local_mem_size,
  error_correction_support,
  host_unified_memory,
  profiling_timer_resolution,
  endian_little,
  is_available,
  is_compiler_available,
  is_linker_available,
  execution_capabilities,
  queue_properties,
  built_in_kernels,
  platform,
  name,
  vendor,
  driver_version,
  profile,
  device_version,
  opencl_version,
  extensions,
  printf_buffer_size,
  preferred_interop_user_sync,
  parent_device,
  partition_max_sub_devices,
  partition_properties,
  partition_affinity_domain,
  partition_type,
  reference_count
};

enum class device_partition_property : int {
  unsupported,
  partition_equally,
  partition_by_counts,
  partition_by_affinity_domain,
  partition_affinity_domain_next_partitionable
};

enum class device_affinity_domain : int {
  unsupported,
  numa,
  L4_cache,
  L3_cache,
  L2_cache,
  next_partitionable
};

enum class device_partition_type : int {
  no_partition,
  numa,
  L4_cache,
  L3_cache,
  L2_cache,
  L1_cache
};

enum class local_mem_type : int {
  none,
  local,
  global
};

enum class fp_config : int {
  denorm,
```

```
  inf_nan,
  round_to_nearest,
  round_to_zero,
  round_to_inf,
  fma,
  correctly_rounded_divide_sqrt,
  soft_float
};

enum class global_mem_cache_type : int {
  none,
  read_only,
  write_only
};

enum class device_execution_capabilities : unsigned int {
  exec_kernel,
  exec_native_kernel
};


using device_fp_config = unsigned int;
using device_exec_capabilities = unsigned int;
using device_queue_properties = unsigned int;


/** Query the return type for get_info() on context stuff

    \todo To be implemented, return always void.
*/
TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::device, void)

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_INFO_DEVICE_HPP
```

11.73 include/CL/sycl/device/detail/device_tail.hpp File Reference

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.74 device_tail.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP

/** \file The ending part of of OpenCL SYCL device

    This is here to break a dependence between device and device_selector

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Return a list of all available devices

    Return synchronous errors via SYCL exception classes.
*/
vector_class<device>
device::get_devices(info::device_type device_type) {
  // Start with the default device
  vector_class<device> devices = { {} };

#ifdef TRISYCL_OPENCL
  // Then add all the OpenCL devices
  for (const auto &d : boost::compute::system::devices())
    devices.emplace_back(d);
#endif

  // The selected devices
  vector_class<device> sd;
  device_type_selector s { device_type };

  // Return the devices with the good criterion according to the selector
  std::copy_if(devices.begin(), devices.end(), std::back_inserter(sd),
               [&](const device &e) { return s(e) >= 0; });
  return sd;
}

/// @} to end the Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_DEVICE_TAIL_HPP
```

11.75 include/CL/sycl/device/detail/host_device.hpp File Reference

```
#include <memory>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
```

```
#include "CL/sycl/platform.hpp"
```

Include dependency graph for host_device.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::host_device

  SYCL host device.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.76 host_device.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP

/** \file The OpenCL SYCL host device implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** SYCL host device

    \todo The implementation is quite minimal for now. :-)
*/
class host_device : public detail::device,
                    public detail::singleton<host_device> {

public:

#ifdef TRISYCL_OPENCL
  /** Return the cl_device_id of the underlying OpenCL platform

      This throws an error since there is no OpenCL device associated
      to the host device.
  */
  cl_device_id get() const override {
    throw non_cl_error("The host device has no OpenCL device");
  }


  /** Return the underlying Boost.Compute device

      This throws an error since there is no OpenCL device associated
      to the host device.
  */
  boost::compute::device &get_boost_compute() override {
    throw non_cl_error("The host device has no underlying OpenCL device");
  }
#endif


  /// Return true since the device is a SYCL host device
  bool is_host() const override {
    return true;
  }


  /// Return false since the host device is not an OpenCL CPU device
  bool is_cpu() const override {
    return false;
  }


  /// Return false since the host device is not an OpenCL GPU device
  bool is_gpu() const override {
    return false;
  }


  /// Return false since the host device is not an OpenCL accelerator device
  bool is_accelerator() const override {
    return false;
  }


  /** Return the platform of device

      Return synchronous errors via the SYCL exception class.
```

```
00085
00086       \todo To be implemented
00087   */
00088   cl::sycl::platform get_platform() const override {
00089     detail::unimplemented();
00090     return {};
00091   }
00092
00093 #if 0
00094   /** Query the device for OpenCL info::device info
00095
00096       Return synchronous errors via the SYCL exception class.
00097
00098       \todo To be implemented
00099   */
00100   template <info::device Param>
00101   typename info::param_traits<info::device, Param>::type
00102   get_info() const override {
00103     detail::unimplemented();
00104     return {};
00105   }
00106 #endif
00107
00108   /** Specify whether a specific extension is supported on the device
00109
00110       \todo To be implemented
00111   */
00112   bool has_extension(const string_class &extension) const override {
00113     detail::unimplemented();
00114     return {};
00115   }
00116
00117
00118 };
00119
00120 /// @} to end the execution Doxygen group
00121
00122 }
00123 }
00124 }
00125
00126 /*
00127     # Some Emacs stuff:
00128     ### Local Variables:
00129     ### ispell-local-dictionary: "american"
00130     ### eval: (flyspell-prog-mode)
00131     ### End:
00132 */
00133
00134 #endif // TRISYCL_SYCL_DEVICE_DETAIL_HOST_DEVICE_HPP
```

11.77 include/CL/sycl/device/detail/opencl_device.hpp File Reference

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for opencl_device.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::opencl_device

  SYCL OpenCL device.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.78 opencl_device.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP
#define TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP

/** \file The SYCL OpenCL device implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

```

```
#include <memory>

#include <boost/compute.hpp>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device/detail/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/// SYCL OpenCL device
class opencl_device : public detail::device {

  /// Use the Boost Compute abstraction of the OpenCL device
  boost::compute::device d;

  /** A cache to always return the same alive device for a given
      OpenCL device

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_device_id, detail::opencl_device>
 cache;

public:

  /// Return the cl_device_id of the underlying OpenCL device
  cl_device_id get() const override {
    return d.id();
  }


  /// Return the underlying Boost.Compute device
  boost::compute::device &get_boost_compute() override {
    return d;
  }


  /// Return false since an OpenCL device is not the SYCL host device
  bool is_host() const override {
    return false;
  }


  /// Test if the OpenCL is a CPU device
  bool is_cpu() const override {
    // Even in Boost.Compute the type is a bit-field, so use & instead of ==
    return d.type() & boost::compute::device::cpu;
  }


  /// Test if the OpenCL is a GPU device
  bool is_gpu() const override {
    // Even in Boost.Compute the type is a bit-field, so use & instead of ==
    return d.type() & boost::compute::device::gpu;
  }


  /// Test if the OpenCL is an accelerator device
  bool is_accelerator() const override {
    // Even in Boost.Compute the type is a bit-field, so use & instead of ==
    return d.type() & boost::compute::device::accelerator;
  }


  /** Return the platform of device

      Return synchronous errors via the SYCL exception class.
  */
  cl::sycl::platform get_platform() const override {
    return d.platform();
  }

#if 0
  /** Query the device for OpenCL info::device info

      Return synchronous errors via the SYCL exception class.

      \todo To be implemented
  */
```

```
  template <info::device Param>
  typename info::param_traits<info::device, Param>::type
  get_info() const override {
    detail::unimplemented();
    return {};
  }
#endif

  /** Specify whether a specific extension is supported on the device.

      \todo To be implemented
  */
  bool has_extension(const string_class &extension) const override {
    detail::unimplemented();
    return {};
  }


  ///// Get a singleton instance of the opencl_device
  static std::shared_ptr<opencl_device>
  instance(const boost::compute::device &d) {
    return cache.get_or_register(d.id(),
                                 [&] { return new opencl_device { d }; });
  }

private:

  /// Only the instance factory can build it
  opencl_device(const boost::compute::device &d) : d { d } {}

public:

  /// Unregister from the cache on destruction
  ~opencl_device() override {
    cache.remove(d.id());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
TRISYCL_WEAK_ATTRIB_PREFIX
detail::cache<cl_device_id, detail::opencl_device>
opencl_device::cache
TRISYCL_WEAK_ATTRIB_SUFFIX;

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_DETAIL_OPENCL_DEVICE_HPP
```

11.79 include/CL/sycl/device_runtime.hpp File Reference

```
#include <algorithm>
#include <map>
#include <memory>
#include <vector>
#include <boost/compute.hpp>
#include <boost/optional.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#include "CL/sycl/device/detail/opencl_device.hpp"
```

```
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for device_runtime.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::drt::accessor< Accessor >

  SYCL accessor seen from a device perspective. More...
- struct cl::sycl::drt::code::program

  A binary program containing some kernels.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::drt

  The device-side runtime.
- cl::sycl::drt::code

  The binary code of the kernels.

Functions

- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::serialize_arg (detail::task &task, std::size_t index, void *arg, std::size_t arg_size)

  Set an argument value of the kernel.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::serialize_accessor_arg (detail::task &task, std::size_t index, void *arg, std::size_t arg_size)

  Set an accessor argument of the kernel.
- TRISYCL_WEAK_ATTRIB_PREFIX void TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::set_kernel (detail::task &task, const char *kernel_name, const char *kernel_short_name)

  Set the kernel for a later invocation.

Variables

- TRISYCL_WEAK_ATTRIB_PREFIX std::map< std::string, std::vector< unsigned char > > TRISYCL_WEAK_ATTRIB_SUFFIX cl::sycl::drt::kernel_IR

  The code of the kernels arranged by name.

11.80 device_runtime.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_RUNTIME_HPP
#define TRISYCL_SYCL_DEVICE_RUNTIME_HPP

/** \file The OpenCL SYCL device-side runtime

    This code is expected to be called once some Clang & LLVM passes
    of the device compiler has been applied.

    \todo Refactor to be more modular and OpenCL-independent so it can
    more easily be retargeted to some other runtime API.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <map>
#include <memory>
#include <vector>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif
#include <boost/optional.hpp>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/device/detail/host_device.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/device/detail/opencl_device.hpp"
#endif
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {

/** \addtogroup device_runtime Device-side runtime implementation
    @{
*/

/// The device-side runtime
namespace drt {

```

```
/// SYCL accessor seen from a device perspective
template <typename Accessor>
class accessor :
    public detail::container_element_aspect<typename Accessor::value_type>
 {

  /** The pointer to the data

      For now none of the following works yet for various reasons:
      \code
      cl::sycl::global_ptr<typename Accessor::value_type> buffer;
      cl::sycl::global<typename Accessor::value_type *> buffer;
     __global typename Accessor::pointer buffer;
     \endcode

     So use a more trivial approach
  */
  TRISYCL_GLOBAL_AS typename Accessor::value_type *buffer;

  /** The size of the accessor

      The compiler should optimize it out if it is unused */
  std::size_t size;

public:

  /// Capture the minimal accessor features
  accessor(Accessor &a) :
    /* Initialize the buffer with actually a \c cl_mem since it will be
       overridden with the kernel argument by the serializer at
       run-time, but at least from the kernel point of view it has to
       look like an \c Accessor::value_type * for the kernel
       outlining */
    buffer {
      reinterpret_cast<TRISYCL_GLOBAL_AS typename Accessor::value_type *>
        (a.implementation->get_order())
        },
    size { a.get_count() }
  {
#ifndef TRISYCL_DEVICE
    /* Register the buffer address to be updated with the final
       version before the kernel set arg */
    TRISYCL_DUMP_T("drt::accessor(Accessor &a) : &a = "
                   << (void *) &a << '\n'
                   << "\t&buffer = " << (void *) &buffer
                   << ", Order = " << a.implementation->get_order());
#endif
  }


  /// use the accessor with integers à la []
  auto &operator[](std::size_t index) const noexcept {
    return buffer[index];
  }

};


/** Set an argument value of the kernel

    \param[in] task is the implementation detail of a triSYCL kernel

    \param[in] index is the order of the argument to set (0 is the first one)

    \param[in] arg points to the argument value

    \param[in] arg_size is the size of the argument
*/
TRISYCL_WEAK_ATTRIB_PREFIX void
TRISYCL_WEAK_ATTRIB_SUFFIX
serialize_arg(detail::task &task,
              std::size_t index,
              void *arg,
              std::size_t arg_size) {
  TRISYCL_DUMP_T("serialize_arg index = " << index
                 << ", size = " << arg_size << ", arg = " << arg);
  task.set_arg(index, arg_size, arg);
}


/** Set an accessor argument of the kernel

    \param[in] task is the implementation detail of a triSYCL kernel

    \param[in] index is the order of the argument to set (0 is the first one)

    \param[in] arg points to the argument value
```

```

    \param[in] arg_size is the size of the argument
*/
TRISYCL_WEAK_ATTRIB_PREFIX void
TRISYCL_WEAK_ATTRIB_SUFFIX
serialize_accessor_arg(detail::task &task,
                       std::size_t index,
                       void *arg,
                       std::size_t arg_size) {
  TRISYCL_DUMP_T("serialize_accessor_arg index = " << index
                 << ", size = " << arg_size << ", arg = " << arg);
#ifdef TRISYCL_OPENCL
  /* The address of the argument has been actually initialized by the
     accessor constructor to the accessor order to relate an argument
     to its accessor */
  task.set_arg(index,
               task.get_compute_buffer(reinterpret_cast<std::size_t>(arg)));
#endif
}


/// The binary code of the kernels
namespace code {

  /// A binary program containing some kernels
  struct program {
    /// The size of a binary program
    std::size_t binary_size;
    /// The bytes of program. Use this type for \c boost::compute
    unsigned const char *binary;

    /** The current program

        \todo Deal with several programs
    */
    static boost::optional<program> p;

    program(std::size_t binary_size, const char *binary)
      : binary_size { binary_size }
      , binary { reinterpret_cast<unsigned const char *>(binary) } {
        p = *this;
        TRISYCL_DUMP_T("Create program with binary size = 0x" << binary_size);
      }
  };

  TRISYCL_WEAK_ATTRIB_PREFIX boost::optional<program>
  TRISYCL_WEAK_ATTRIB_SUFFIX program::p;

}

/** The code of the kernels arranged by name

  \todo use std::byte when moving to C++17
*/
TRISYCL_WEAK_ATTRIB_PREFIX std::map<std::string, std::vector<unsigned char>>
TRISYCL_WEAK_ATTRIB_SUFFIX kernel_IR;


/** Set the kernel for a later invocation

    \param[in] task is the implementation detail of a triSYCL kernel

    \param[in] kernel_name is set by the device compiler to identify
    the kernel to call
*/
TRISYCL_WEAK_ATTRIB_PREFIX void
TRISYCL_WEAK_ATTRIB_SUFFIX
set_kernel(detail::task &task,
           const char *kernel_name,
           const char *kernel_short_name) {
  TRISYCL_DUMP_T("set_kernel setting up " << kernel_name
                 << "\n\taka " << kernel_short_name);
  // \todo Add kernel caching per device
  // auto binary = kernel_IR.find(kernel_name);
  // if (binary == kernel_IR.end()) {
#ifdef TRISYCL_OPENCL
  auto context = task.get_queue()->get_boost_compute().get_context();
  // Construct an OpenCL program from the precompiled kernel file
  auto
#ifdef SDX_KERNEL_PROGRAM_OWNING_BUG
    /* There is a bug in Xilinx OpenCL runtime where the cl_kernel
       does not keep alive its own cl_program */
    static
#endif
    program = boost::compute::program::create_with_binary
    (code::program::p->binary,
     code::program::p->binary_size,
```

```
    context);
  TRISYCL_DUMP_T("...on device with name "
                 << task.get_queue()->get_boost_compute().get_device().name());
  // Build the OpenCL program
  program.build();

  // Build a SYCL kernel from the OpenCL kernel
  cl::sycl::kernel k { boost::compute::kernel { program, kernel_short_name } };

  task.set_kernel(k.implementation);
#endif
}

/// @} to end the Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_RUNTIME_HPP
```

11.81 include/CL/sycl/device_selector.hpp File Reference

```
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
```

Include dependency graph for device_selector.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::device_selector

  The SYCL heuristics to select a device. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.82 device_selector.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_SELECTOR_HPP
#define TRISYCL_SYCL_DEVICE_SELECTOR_HPP

/** \file The OpenCL SYCL device_selector

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/unimplemented.hpp"
```

```
#include "CL/sycl/device.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** The SYCL heuristics to select a device

    The device with the highest score is selected
*/
class device_selector {

public:

  /** Returns a selected device using the functor operator defined in
      sub-classes operator()(const device &dev)

      \todo Remove this from specification
  */
  void /* device */ select_device() const {
    //    return {};
  }


  /**  This pure virtual operator allows the customization of device
       selection.

       It defines the behavior of the device_selector functor called by
       the SYCL runtime on device selection. It returns a "score" for each
       device in the system and the highest rated device will be used
       by the SYCL runtime.
  */
  virtual int operator()(const device &dev) const = 0;


  /// Virtual destructor so the final destructor can be called if any
  virtual ~device_selector() {}

};

/// @} to end the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_SELECTOR_HPP
```

11.83 include/CL/sycl/device_selector/detail/device_selector_tail.hpp File Reference

```
#include <boost/compute.hpp>
```

Include dependency graph for device_selector_tail.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::device_type_selector

  A device selector by device_type. More...
- class cl::sycl::device_typename_selector< DeviceType >

  Select a device by template device_type parameter. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- using cl::sycl::default_selector = device_typename_selector< info::device_type::automatic >

  Devices selected by heuristics of the system.
- using cl::sycl::gpu_selector = device_typename_selector< info::device_type::gpu >

  Select devices according to device type info::device::device_type::gpu from all the available OpenCL devices.
- using cl::sycl::cpu_selector = device_typename_selector< info::device_type::cpu >

  Select devices according to device type info::device::device_type::cpu from all the available devices and heuristics.
- using cl::sycl::host_selector = device_typename_selector< info::device_type::host >

  Selects the SYCL host CPU device that does not require an OpenCL runtime.

11.84 device_selector_tail.hpp

```
#ifndef TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
#define TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP

/** \file The ending part of of the OpenCL SYCL device_selector

    This is here to break a dependence between device and device_selector

    \todo Implement lacking SYCL 2.2 selectors

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/


/** A device selector by device_type

    \todo To be added to the specification
*/
class device_type_selector : public device_selector {

private:

  /// The device_type to select
  info::device_type device_type;

  /** Cache the default device to select with the default device
      selector.

      This is the host device at construction time and remains as is
      if there is no openCL device */
  device default_device;

public:

  device_type_selector(info::device_type device_type)
    : device_type { device_type } {
    // The default device selection heuristic
#ifdef TRISYCL_OPENCL
    if (device_type == info::device_type::automatic) {
      // Ask Boost.Compute for the default OpenCL device
      try {
        default_device = boost::compute::system::default_device();
      }
      catch (...) {
        /* If there is no OpenCL device, just keep the
           default-constructed device, which is the host device */
      }
    }
#endif
```

```
  }


  // To select only the requested device_type
  int operator()(const device &dev) const override {
    if (device_type == info::device_type::all)
      // All devices fit all
      return 1;

    if (device_type == info::device_type::automatic)
      // Only select the default device
      return dev == default_device ? 1 : -1;

    if (device_type == info::device_type::opencl)
      // For now, any non host device is an OpenCL device
      return dev.is_host() ? -1 : 1;

    return dev.type() == device_type ? 1 : -1;
  }

};


/** Select a device by template device_type parameter

    \todo To be added to the specification
*/
template <info::device_type DeviceType>
class device_typename_selector : public device_type_selector {

public:

  device_typename_selector() : device_type_selector { DeviceType } {}

};


/** Devices selected by heuristics of the system

    If no OpenCL device is found then it defaults to the SYCL host device.

    To influence the default device selection, use the Boost.Compute
    environment variables:

    - \c BOOST_COMPUTE_DEFAULT_DEVICE

    - \c BOOST_COMPUTE_DEFAULT_DEVICE_TYPE

    - \c BOOST_COMPUTE_DEFAULT_PLATFORM

    - \c BOOST_COMPUTE_DEFAULT_VENDOR
*/
using default_selector = device_typename_selector<info::device_type::automatic>;


  /** Select devices according to device type info::device::device_type::gpu
    from all the available OpenCL devices.

    If no OpenCL GPU device is found the selector fails.

    Select the best GPU, if any.
*/
using gpu_selector = device_typename_selector<info::device_type::gpu>;


/** Select devices according to device type info::device::device_type::cpu
    from all the available devices and heuristics

    If no OpenCL CPU device is found the selector fails.
*/
using cpu_selector = device_typename_selector<info::device_type::cpu>;


/** Selects the SYCL host CPU device that does not require an OpenCL
    runtime
*/
using host_selector = device_typename_selector<info::device_type::host>;

/// @} to end the execution Doxygen group

```

```
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DEVICE_SELECTOR_DETAIL_DEVICE_SELECTOR_TAIL_HPP
```

11.85 include/CL/sycl/error_handler.hpp File Reference

`#include "CL/sycl/exception.hpp"`

Include dependency graph for error_handler.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::error_handler

  User supplied error handler to call a user-provided function when an error happens from a SYCL object that was constructed with this error handler. More...
- struct cl::sycl::trisycl::default_error_handler

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::trisycl

11.86 error_handler.hpp

```
#ifndef TRISYCL_SYCL_ERROR_HANDLER_HPP
#define TRISYCL_SYCL_ERROR_HANDLER_HPP

/** \file The OpenCL SYCL error_handler

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/exception.hpp"

namespace cl {
namespace sycl {

/** \addtogroup error_handling Error handling
    @{
*/

/// \todo Refactor when updating to latest specification
namespace trisycl {
  // Create a default error handler to be used when nothing is specified
  struct default_error_handler;
}


/** User supplied error handler to call a user-provided function when an
    error happens from a SYCL object that was constructed with this error
    handler
*/
struct error_handler {
  /** The method to define to be called in the case of an error

      \todo Add "virtual void" to the specification
  */
  virtual void report_error(exception &error) = 0;

  /** Add a default_handler to be used by default

      \todo add this concept to the specification?
  */
  static trisycl::default_error_handler
default_handler;


  virtual ~error_handler() = 0;
};


namespace trisycl {

  struct default_error_handler : error_handler {

    void report_error(exception &) override {
    }
  };
```

```
}

  // \todo finish initialization
  //error_handler::default_handler = nullptr;


/// @} End the error_handling Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ERROR_HANDLER_HPP
```

11.87 include/CL/sycl/event.hpp File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::event

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.88 event.hpp

```
#ifndef TRISYCL_SYCL_EVENT_HPP
#define TRISYCL_SYCL_EVENT_HPP

/** \file The event class

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/
namespace cl {
namespace sycl {

class event {

public:

  event() = default;


/** \todo To be implemented */
#if 0
  explicit event(cl_event clEvent);

  event(const event & rhs);

  cl_event get();

  vector_class<event> get_wait_list();

  void wait();

  static void wait(const vector_class<event> &eventList);

  void wait_and_throw();

  static void wait_and_throw(const vector_class<event> &eventList);

  template <info::event param>
  typename param_traits<info::event, param>::type get_info() const;

  template <info::event_profiling param>
  typename param_traits<info::event_profiling,
                        param>::type get_profiling_info() const;
#endif
};

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_EVENT_HPP
```

11.89 include/CL/sycl/exception.hpp File Reference

```
#include <exception>
```

Include dependency graph for exception.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::exception_list

  Exception list to store several exceptions. More...
- class cl::sycl::exception

  Encapsulate a SYCL error information. More...
- class cl::sycl::cl_exception

  Returns the OpenCL error code encapsulated in the exception. More...
- struct cl::sycl::async_exception

  An error stored in an exception_list for asynchronous errors. More...
- class cl::sycl::runtime_error

- class cl::sycl::kernel_error

  Error that occurred before or while enqueuing the SYCL kernel. More...
- class cl::sycl::accessor_error

  Error regarding the cl::sycl::accessor objects defined. More...
- class cl::sycl::nd_range_error

  Error regarding the cl::sycl::nd_range specified for the SYCL kernel. More...
- class cl::sycl::event_error

  Error regarding associated cl::sycl::event objects. More...
- class cl::sycl::invalid_parameter_error

  Error regarding parameters to the SYCL kernel, it may apply to any captured parameters to the kernel lambda. More...
- class cl::sycl::device_error

  The SYCL device will trigger this exception on error. More...
- class cl::sycl::compile_program_error

  Error while compiling the SYCL kernel to a SYCL device. More...
- class cl::sycl::link_program_error

  Error while linking the SYCL kernel to a SYCL device. More...
- class cl::sycl::invalid_object_error

  Error regarding any memory objects being used inside the kernel. More...
- class cl::sycl::memory_allocation_error

  Error on memory allocation on the SYCL device for a SYCL kernel. More...
- class cl::sycl::pipe_error

  A failing pipe error will trigger this exception on error. More...
- class cl::sycl::platform_error

  The SYCL platform will trigger this exception on error. More...
- class cl::sycl::profiling_error

  The SYCL runtime will trigger this error if there is an error when profiling info is enabled. More...
- class cl::sycl::feature_not_supported

  Exception thrown when an optional feature or extension is used in a kernel but its not available on the device the SYCL kernel is being enqueued on. More...
- class cl::sycl::non_cl_error

  Exception for an OpenCL operation requested in a non OpenCL area. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Typedefs

- using cl::sycl::exception_ptr = std::exception_ptr

  A shared pointer to an exception as in C++ specification.
- using cl::sycl::async_handler = function_class< void, exception_list >

11.90 exception.hpp

```
#ifndef TRISYCL_SYCL_EXCEPTION_HPP
#define TRISYCL_SYCL_EXCEPTION_HPP

/** \file The OpenCL SYCL exception

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <exception>

namespace cl {
namespace sycl {

/** \addtogroup error_handling Error handling
    @{
*/


/** A shared pointer to an exception as in C++ specification

    \todo Do we need this instead of reusing directly the one from C++11?
*/
using exception_ptr = std::exception_ptr;


/** Exception list to store several exceptions

    \todo Do we need to define it in SYCL or can we rely on plain C++17 one?
*/
struct exception_list : std::vector<exception_ptr> {
  using std::vector<exception_ptr>::vector;
};

using async_handler = function_class<void, exception_list>
;


/// Encapsulate a SYCL error information
class exception {

  /// The error message to return
  string_class message;

public:

  /// Construct an exception with a message for internal use
  exception(const string_class &message) : message { message } {}

  /// Returns a descriptive string for the error, if available
  string_class what() const {
    return message;
  }


  /** Returns the context that caused the error

      Returns nullptr if not a buffer error.

      \todo Cannot return nullptr. Use optional? Use a specific exception type?
  */
  //context get_context()

};


/// Returns the OpenCL error code encapsulated in the exception
class cl_exception : public exception {

#ifdef TRISYCL_OPENCL
  /// The OpenCL error code to return

  cl_int cl_code;

public:

  /**  Construct an exception with a message and OpenCL error code for
       internal use */
  cl_exception(const string_class &message, cl_int cl_code)
    : exception { message }, cl_code { cl_code } {}

  // thrown as a result of an OpenCL API error code
```

```
  cl_int get_cl_code() const {
    return cl_code;
  }
#endif

};


/// An error stored in an exception_list for asynchronous errors
struct async_exception : exception {
  using exception::exception;
};


class runtime_error : public exception {
  using exception::exception;
};


/// Error that occurred before or while enqueuing the SYCL kernel
class kernel_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// Error regarding the cl::sycl::accessor objects defined
class accessor_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// Error regarding the cl::sycl::nd_range specified for the SYCL kernel
class nd_range_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// Error regarding associated cl::sycl::event objects
class event_error : public runtime_error {
  using runtime_error::runtime_error;
};


/** Error regarding parameters to the SYCL kernel, it may apply to any
    captured parameters to the kernel lambda
*/
class invalid_parameter_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// The SYCL device will trigger this exception on error
class device_error : public exception {
  using exception::exception;
};


/// Error while compiling the SYCL kernel to a SYCL device
class compile_program_error : public device_error {
  using device_error::device_error;
};


/// Error while linking the SYCL kernel to a SYCL device
class link_program_error : public device_error {
  using device_error::device_error;
};


/// Error regarding any memory objects being used inside the kernel
class invalid_object_error : public device_error {
  using device_error::device_error;
};


/// Error on memory allocation on the SYCL device for a SYCL kernel
class memory_allocation_error : public device_error {
  using device_error::device_error;
};


/// A failing pipe error will trigger this exception on error
class pipe_error : public runtime_error {
  using runtime_error::runtime_error;
};


```

```
/// The SYCL platform will trigger this exception on error
class platform_error : public device_error {
  using device_error::device_error;
};


/** The SYCL runtime will trigger this error if there is an error when
    profiling info is enabled
*/
class profiling_error : public device_error {
  using device_error::device_error;
};


/** Exception thrown when an optional feature or extension is used in
    a kernel but its not available on the device the SYCL kernel is
    being enqueued on
*/
class feature_not_supported : public device_error {
  using device_error::device_error;
};


/** Exception for an OpenCL operation requested in a non OpenCL area

    \todo Add to the specification

    \todo Clean implementation

    \todo Exceptions are named error in C++
*/
class non_cl_error : public runtime_error {
  using runtime_error::runtime_error;
};


/// @} End the error_handling Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_EXCEPTION_HPP
```

11.91 include/CL/sycl/group.hpp File Reference

```
#include <cstddef>
#include <functional>
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for group.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::group< Dimensions >

  A group index used in a parallel_for_workitem to specify a work_group. More...
- struct cl::sycl::group< Dimensions >

  A group index used in a parallel_for_workitem to specify a work_group. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.

11.92 group.hpp

```
#ifndef TRISYCL_SYCL_GROUP_HPP
#define TRISYCL_SYCL_GROUP_HPP

/** \file The OpenCL SYCL nd_item<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <functional>

#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

template <int Dimensions = 1>
struct group;

namespace detail {

template <int Dimensions = 1, typename ParallelForFunctor>
void parallel_for_workitem(const group<Dimensions> &g,
                           ParallelForFunctor f);

}

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A group index used in a parallel_for_workitem to specify a work_group
 */
template <int Dimensions>
struct group {
  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = Dimensions;

private:

  /// The coordinate of the group item
  id<Dimensions> group_id;

  /// Keep a reference on the nd_range to serve potential query on it
  nd_range<Dimensions> ndr;

public:

  /** Create a group from an nd_range<> with a 0 id<>

      \todo This should be private since it is only used by the triSYCL
      implementation
  */
  group(const nd_range<Dimensions> &ndr) : ndr { ndr } {}


  /** Create a group from an id and a nd_range<>

      \todo This should be private somehow, but it is used by the
      validation infrastructure
  */
  group(const id<Dimensions> &i, const nd_range<Dimensions> &ndr )
    :
    group_id { i }, ndr { ndr } {}
```

```
/** To be able to copy and assign group, use default constructors too

    \todo Make most of them protected, reserved to implementation
*/
group() = default;


/** Return an id representing the index of the group within the nd_range
    for every dimension
*/
id<Dimensions> get_id() const { return group_id; }


/// Return the index of the group in the given dimension
size_t get_id(int dimension) const { return get_id()[dimension]; }


/** Return the index of the group in the given dimension within the
    nd_range<>

    \todo In this implementation it is not const because the group<> is
    written in the parallel_for iterators. To fix according to the
    specification
*/
auto &operator[](int dimension) {
  return group_id[dimension];
}


/** Return a range<> representing the dimensions of the current
    group

    This local range may have been provided by the programmer, or chosen
    by the runtime.

    \todo Fix this comment and the specification
*/
range<Dimensions> get_group_range() const {
  return get_nd_range().get_group();
}


/// Return element dimension from the con stituent group range
size_t get_group_range(int dimension) const {
  return get_group_range()[dimension];
}


/// Get the local range for this work_group
range<Dimensions> get_global_range() const {
  return get_nd_range().get_global();
}


/// Return element dimension from the constituent global range
size_t get_global_range(int dimension) const {
  return get_global_range()[dimension];
}


/** Get the local range for this work_group

    \todo Add to the specification
*/
range<Dimensions> get_local_range() const {
  return get_nd_range().get_local();
}


/** Return element dimension from the constituent local range

    \todo Add to the specification
*/
size_t get_local_range(int dimension) const {
  return get_local_range()[dimension];
}


/**  Get the offset of the NDRange

     \todo Add to the specification
*/
id<Dimensions> get_offset() const { return get_nd_range().get_offset(); }


```

```
  /**  Get the offset of the NDRange

       \todo Add to the specification
  */
  size_t get_offset(int dimension) const { return get_offset()[dimension]; }


  /// \todo Also provide this access to the current nd_range
  nd_range<Dimensions> get_nd_range() const { return ndr; }


  /** Get a linearized version of the group ID

   */
  size_t get_linear() const {
    return detail::linear_id(get_group_range(), get_id());
  }


  /** Loop on the work-items inside a work-group

      \todo Add this method in the specification
  */
  void parallel_for_work_item(std::function<void(nd_item<dimensionality>)> f)
    const {
    detail::parallel_for_workitem(*this, f);
  }


  /** Loop on the work-items inside a work-group

      \todo Add this method in the specification
  */
  void parallel_for_work_item(std::function<void(item<dimensionality>)> f)
    const {
    auto item_adapter = [=] (nd_item<dimensionality> ndi) {
      item<dimensionality> i = ndi.get_item();
      f(i);
    };
    detail::parallel_for_workitem(*this, item_adapter);
  }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_GROUP_HPP
```

11.93 include/CL/sycl/handler.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <tuple>
#include <utility>
#include <boost/compute.hpp>
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/instantiate_kernel.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/opencl_types.hpp"
```

```
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for handler.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::handler

  Command group handler class. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_parallel_for_functor_GLOBAL(N)

  SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>
- #define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)
- #define TRISYCL_ParallelForKernel_RANGE(N)

  Kernel invocation method of a kernel defined as a kernel object, for the specified range and given an id or item for indexing in the indexing space defined by range, described in detail in 5.4.

Functions

- static std::shared_ptr< detail::task > cl::sycl::detail::add_buffer_to_task (handler *command_group_handler, std::shared_ptr< detail::buffer_base > b, bool is_write_mode)

  Register a buffer as used by a task.

11.93.1 Macro Definition Documentation

11.93.1.1 TRISYCL_parallel_for_functor_GLOBAL

```
#define TRISYCL_parallel_for_functor_GLOBAL(
            N )
```

Value:

```
template <typename KernelName = std::nullptr_t,                          \
          typename ParallelForFunctor>                                    \
  void parallel_for(range<N> global_size,                                 \
                    ParallelForFunctor f) {                               \
    schedule_kernel<KernelName>([=] {                                     \
        detail::parallel_for(global_size, f);                             \
      });                                                                 \
  }
```

SYCL parallel_for launches a data parallel computation with parallelism specified at launch time by a range<>

Kernel invocation method of a kernel defined as a lambda or functor, for the specified range and given an id or item for indexing in the indexing space defined by range.

If it is a lambda function or the if the functor type is globally visible there is no need for the developer to provide a kernel name type (typename KernelName) for it, as described in detail in 3.5.3

Parameters

global_size	is the full size of the range<>
N	dimensionality of the iteration space
f	is the kernel functor to execute
KernelName	is a class type that defines the name to be used for the underlying kernel

Unfortunately, to have implicit conversion to work on the range, the function can not be templated, so instantiate it for all the Dimensions

Definition at line 236 of file handler.hpp.

11.93.1.2 TRISYCL_ParallelForFunctor_GLOBAL_OFFSET

```
#define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(
            N )
```

Value:

```
template <typename KernelName = std::nullptr_t,                  \
            typename ParallelForFunctor>                          \
  void parallel_for(range<N> global_size,                         \
                    id<N> offset,                                 \
                    ParallelForFunctor f) {                       \
    schedule_kernel<KernelName>([=] {                             \
        detail::parallel_for_global_offset(global_size,
       \
                                           offset,                \
                                           f);                    \
      });                                                         \
  }
```

11.93.1.3 TRISYCL_ParallelForKernel_RANGE

```
#define TRISYCL_ParallelForKernel_RANGE(
               N )
```

Value:

```
void parallel_for(range<N> num_work_items,                          \
                  kernel sycl_kernel) {                              \
    /* For now just use the usual host task system to schedule      \
       manually the OpenCL kernels instead of using OpenCL event-based \
       scheduling                                                   \
                                                                    \
       \todo Move the tracing inside the kernel implementation      \
                                                                    \
       \todo Simplify this 2 step ugly interface                    \
    */                                                              \
    task->set_kernel(sycl_kernel.implementation);                   \
    /* Use an intermediate variable to capture task by copy because \
       otherwise "this" is captured by reference and havoc with task \
       just accessing the dead "this". Nasty bug to find... */      \
    task->schedule(detail::trace_kernel<kernel>([=, t = task] {     \
          sycl_kernel.implementation->parallel_for(t, t->get_queue(), \
                                                   num_work_items); })); \
  }
```

Kernel invocation method of a kernel defined as a kernel object, for the specified range and given an id or item for indexing in the indexing space defined by range, described in detail in 5.4.

Todo Add in the spec a version taking a kernel and a functor, to have host fall-back

Definition at line 413 of file handler.hpp.

11.94 handler.hpp

```
00001 #ifndef TRISYCL_SYCL_HANDLER_HPP
00002 #define TRISYCL_SYCL_HANDLER_HPP
00003
00004 /** \file The OpenCL SYCL command group handler
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013 #include <memory>
00014 #include <tuple>
00015 #include <utility>
```

```

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/accessor.hpp"
#include "CL/sycl/command_group/detail/task.hpp"
#include "CL/sycl/detail/instantiate_kernel.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/kernel.hpp"
#include "CL/sycl/opencl_types.hpp"
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Command group handler class

    A command group handler object can only be constructed by the SYCL runtime.

    All of the accessors defined in the command group scope take as a
    parameter an instance of the command group handler and all the kernel
    invocation functions are methods of this class.
*/
class handler : public detail::debug<handler> {

public:

  /** Attach the task and accessors to it.
   */
  std::shared_ptr<detail::task> task;


  /* Create a command group handler from the queue detail

     The queue detail is used to track kernel completion.

     Note that this is an implementation dependent constructor. Normal
     users cannot construct handler from scratch.

     \todo Make this constructor private
  */
  handler(const std::shared_ptr<detail::queue> &q) {
    // Create a new task for this command_group
    task = std::make_shared<detail::task>(q);
  }


#ifdef TRISYCL_OPENCL
  /** Set accessor kernel arg for an OpenCL kernel which is used through the
      SYCL/OpenCL interop interface

      The index value specifies which parameter of the OpenCL kernel is
      being set and the accessor object, which OpenCL buffer or image is
      going to be given as kernel argument.

      \todo Update the specification to use a ref && to the accessor instead?

      \todo It is not that clean to have set_arg() associated to a
      command handler. Rethink the specification?

      \todo It seems more logical to have these methods on kernel instead
  */
  template <typename DataType,
            int Dimensions,
            access::mode Mode,
            access::target Target = access::target::global_buffer
>
  void set_arg(int arg_index,
               accessor<DataType, Dimensions, Mode, Target> &&
acc_obj) {
    /* Think about setting the kernel argument before actually calling
       the kernel.

       Explicitly capture task by copy instead of having this captured
       by reference and task by reference by side effect */
    task->add_prelude([=, task = task] {
        task->set_arg(arg_index, acc_obj.implementation->get_cl_buffer());
      });
  }

```

```

  /** Set kernel args for an OpenCL kernel which is used through the
      SYCL/OpenCL interoperability interface with a wrapper type
  */
  template <typename T, typename = std::enable_if_t<is_wrapper<T>::value> >
  void set_arg(int arg_index, T && scalar_value) {
    /* Explicitly capture task by copy instead of having this captured
       by reference and task by reference by side effect */
    task->add_prelude([=, task = task] {
        task->set_arg(arg_index, scalar_value.unwrap());
      });
  }


  /** Set kernel args for an OpenCL kernel which is used through the
      SYCL/OpenCL interoperability interface without a wrapper type
  */
  template <typename T>
  std::enable_if_t<!is_wrapper<T>::value>
  set_arg(int arg_index, T && scalar_value) {
    /* Explicitly capture task by copy instead of having this captured
       by reference and task by reference by side effect */
    task->add_prelude([=, task = task] {
        task->set_arg(arg_index, scalar_value);
      });
  }

private:

  /// Helper to individually call set_arg() for each argument
  template <std::size_t... Is, typename... Ts>
  void dispatch_set_arg(std::index_sequence<Is...>, Ts&&... args) {
    // Use an intermediate tuple to ease individual argument access
    auto &&t = std::make_tuple(std::forward<Ts>(args)...);
    // Dispatch individual set_arg() for each argument
    auto just_to_evaluate = {
      0 /*< At least 1 element to deal with empty set_args() */,
      ( set_arg(Is, std::forward<Ts>(std::get<Is>(t))), 0)...
    };
    // Remove the warning about unused variable
    static_cast<void>(just_to_evaluate);
  }

public:

  /** Set all kernel args for an OpenCL kernel which is used through the
      SYCL/OpenCL interop interface

      \todo Update the specification to add this function according to
      https://cvs.khronos.org/bugzilla/show_bug.cgi?id=15978 proposal
  */
  template <typename... Ts>
  void set_args(Ts &&... args) {
    /* Construct a set of increasing argument index to be able to call
       the real set_arg */
    dispatch_set_arg(std::index_sequence_for<Ts...>{},
                     std::forward<Ts>(args)...);
  }
#endif

private:

  /** Schedule the kernel

      Add a traced version of the kernel in host mode or add the
      kernel in an instantiating function for later extraction by the
      compiler
  */
  template <typename KernelName,
            typename Kernel>
  void schedule_kernel(Kernel k) {
    /* Explicitly capture task by copy instead of having this captured
       by reference and task by reference by side effect */
    task->schedule(detail::trace_kernel<KernelName>([=, t = task] () mutable {
          if (t->owner_queue->is_host())
            k();
          else {
            TRISYCL_DUMP_T("schedule_kernel &k = " << (void *) &k);

            // Structure the kernel code so it can be outlined and called
            detail::launch_device_kernel<KernelName>(*t, k);
            // \todo for now only deal with 1 physical work-item only
            t->get_kernel().single_task(t, t->get_queue());
          }
        }));
  }

```

```
public:

  /** Kernel invocation method of a kernel defined as a lambda or
      functor. If it is a lambda function or the functor type is globally
      visible there is no need for the developer to provide a kernel name type
      (typename KernelName) for it, as described in 3.5.3

      SYCL single_task launches a computation without parallelism at
      launch time.

      \param F specify the kernel to be launched as a single_task

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel
  */
  template <typename KernelName = std::nullptr_t,
            typename ParallelForFunctor>
  void single_task(ParallelForFunctor &&f) {
    TRISYCL_DUMP_T("single_task &f = " << (void *) &f);

    schedule_kernel<KernelName>(std::forward<ParallelForFunctor>(f));
  }


  /** SYCL parallel_for launches a data parallel computation with
      parallelism specified at launch time by a range<>

      Kernel invocation method of a kernel defined as a lambda or functor,
      for the specified range and given an id or item for indexing in the
      indexing space defined by range.

      If it is a lambda function or the if the functor type is globally
      visible there is no need for the developer to provide a kernel name
      type (typename KernelName) for it, as described in detail in 3.5.3

      \param global_size is the full size of the range<>

      \param N dimensionality of the iteration space

      \param f is the kernel functor to execute

      \param KernelName is a class type that defines the name to be used
      for the underlying kernel

      Unfortunately, to have implicit conversion to work on the range, the
      function can not be templated, so instantiate it for all the
      Dimensions
  */
#define TRISYCL_parallel_for_functor_GLOBAL(N)                          \
  template <typename KernelName = std::nullptr_t,                       \
            typename ParallelForFunctor>                                \
  void parallel_for(range<N> global_size,                               \
                    ParallelForFunctor f) {                             \
    schedule_kernel<KernelName>([=] {                                   \
        detail::parallel_for(global_size, f);                           \
      });                                                               \
  }

  TRISYCL_parallel_for_functor_GLOBAL(1)
  TRISYCL_parallel_for_functor_GLOBAL(2)
  TRISYCL_parallel_for_functor_GLOBAL(3)


  /** Kernel invocation method of a kernel defined as a lambda or functor,
      for the specified range and offset and given an id or item for
      indexing in the indexing space defined by range

      If it is a lambda function or the if the functor type is globally
      visible there is no need for the developer to provide a kernel name
      type (typename KernelName) for it, as described in detail in 3.5.3

      \param global_size is the global size of the range<>

      \param offset is the offset to be add to the id<> during iteration

      \param f is the kernel functor to execute

      \param ParallelForFunctor is the kernel functor type

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel

      Unfortunately, to have implicit conversion to work on the range, the
      function can not be templated, so instantiate it for all the
      dimensions
  */
#define TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(N)                \
```

```
template <typename KernelName = std::nullptr_t,                  \
          typename ParallelForFunctor>                           \
void parallel_for(range<N> global_size,                          \
                  id<N> offset,                                  \
                  ParallelForFunctor f) {                        \
  schedule_kernel<KernelName>([=] {                              \
      detail::parallel_for_global_offset(global_size,            \
                                         offset,                 \
                                         f);                     \
    });                                                          \
}

TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(1)
TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(2)
TRISYCL_ParallelForFunctor_GLOBAL_OFFSET(3)


/** Kernel invocation method of a kernel defined as a lambda or functor,
    for the specified nd_range and given an nd_item for indexing in the
    indexing space defined by the nd_range

    If it is a lambda function or the if the functor type is globally
    visible there is no need for the developer to provide a kernel name
    type (typename KernelName) for it, as described in detail in 3.5.3

    \param r defines the iteration space with the work-group layout and
    offset

    \param Dimensions dimensionality of the iteration space

    \param f is the kernel functor to execute

    \param ParallelForFunctor is the kernel functor type

    \param KernelName is a class type that defines the name to be used for
    the underlying kernel
*/
template <typename KernelName = std::nullptr_t,
          int Dimensions,
          typename ParallelForFunctor>
void parallel_for(nd_range<Dimensions> r,
                  ParallelForFunctor f) {
  schedule_kernel<KernelName>([=] { detail::parallel_for(r, f); });
}


/** Hierarchical kernel invocation method of a kernel defined as a
    lambda encoding the body of each work-group to launch

    May contain multiple kernel built-in parallel_for_work_item
    functions representing the execution on each work-item.

    Launch num_work_groups work-groups of runtime-defined
    size. Described in detail in 3.5.3.

    \param r defines the iteration space with the work-group layout and
    offset

    \param Dimensions dimensionality of the iteration space

    \param f is the kernel functor to execute

    \param ParallelForFunctor is the kernel functor type

    \param KernelName is a class type that defines the name to be used for
    the underlying kernel
*/
template <typename KernelName = std::nullptr_t,
          int Dimensions = 1,
          typename ParallelForFunctor>
void parallel_for_work_group(nd_range<Dimensions> r,
                             ParallelForFunctor f) {
  schedule_kernel<KernelName>([=] {
      detail::parallel_for_workgroup(r, f);
    });
}


/** Hierarchical kernel invocation method of a kernel defined as a
    lambda encoding the body of each work-group to launch

    May contain multiple kernel built-in parallel_for_work_item
    functions representing the execution on each work-item.

    Launch num_work_groups work-groups of runtime-defined
    size. Described in detail in 3.5.3.

```

```
      \param r defines the iteration space with the work-group layout and
      offset

      \param Dimensions dimensionality of the iteration space

      \param f is the kernel functor to execute

      \param ParallelForFunctor is the kernel functor type

      \param KernelName is a class type that defines the name to be used for
      the underlying kernel
  */
  template <typename KernelName = std::nullptr_t,
            int Dimensions = 1,
            typename ParallelForFunctor>
  void parallel_for_work_group(range<Dimensions> r1,
range<Dimensions> r2,
                               ParallelForFunctor f) {
    parallel_for_work_group(nd_range<Dimensions> { r1, r2 }, f);
  }


  /** Kernel invocation method of a kernel defined as pointer to a kernel
      object, described in detail in 3.5.3

      \todo Add in the spec a version taking a kernel and a functor,
      to have host fall-back
  */
  void single_task(kernel sycl_kernel) {
    /* For now just use the usual host task system to schedule           \
       manually the OpenCL kernels instead of using OpenCL event-based  \
       scheduling                                                        \
                                                                         \
       \todo Move the tracing inside the kernel implementation           \
                                                                         \
       \todo Simplify this 2 step ugly interface                         \
    */                                                                   \
    task->set_kernel(sycl_kernel.implementation);                        \
    task->schedule(detail::trace_kernel<kernel>([=, t = task] {          \
          sycl_kernel.implementation->single_task(t, t->get_queue());    \
        }));                                                             \
  }


  /** Kernel invocation method of a kernel defined as a kernel object,
      for the specified range and given an id or item for indexing in
      the indexing space defined by range, described in detail in
      5.4.

      \todo Add in the spec a version taking a kernel and a functor,
      to have host fall-back
  */
#define TRISYCL_ParallelForKernel_RANGE(N)                               \
  void parallel_for(range<N> num_work_items,                             \
                    kernel sycl_kernel) {                                \
    /* For now just use the usual host task system to schedule           \
       manually the OpenCL kernels instead of using OpenCL event-based  \
       scheduling                                                        \
                                                                         \
       \todo Move the tracing inside the kernel implementation           \
                                                                         \
       \todo Simplify this 2 step ugly interface                         \
    */                                                                   \
    task->set_kernel(sycl_kernel.implementation);                        \
    /* Use an intermediate variable to capture task by copy because     \
       otherwise "this" is captured by reference and havoc with task     \
       just accessing the dead "this". Nasty bug to find... */           \
    task->schedule(detail::trace_kernel<kernel>([=, t = task] {          \
          sycl_kernel.implementation->parallel_for(t, t->get_queue(),    \
                                                   num_work_items); })); \
  }


  /* Do not use a template parameter since otherwise the parallel_for
     functor is selected instead of this one

     \todo Clean this
  */
  TRISYCL_ParallelForKernel_RANGE(1)
  TRISYCL_ParallelForKernel_RANGE(2)
  TRISYCL_ParallelForKernel_RANGE(3)
#undef TRISYCL_ParallelForKernel_RANGE

  /** Kernel invocation method of a kernel defined as pointer to a kernel
      object, for the specified nd_range and given an nd_item for indexing
      in the indexing space defined by the nd_range, described in detail
      in 3.5.3
```

```

      \todo Add in the spec a version taking a kernel and a functor,
      to have host fall-back

      \todo To be implemented
  */
  template <int Dimensions = 1>
  void parallel_for(nd_range<Dimensions>, kernel syclKernel) {
    detail::unimplemented();
  }

};

namespace detail {

/** Register a buffer as used by a task

    This is a proxy function to avoid complicated type recursion.
*/
static std::shared_ptr<detail::task>
add_buffer_to_task(handler *command_group_handler,
                   std::shared_ptr<detail::buffer_base> b,
                   bool is_write_mode) {
  command_group_handler->task->add_buffer(b, is_write_mode);
  return command_group_handler->task;
}

}

/// @} End the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_HANDLER_HPP
```

11.95 include/CL/sycl/handler_event.hpp File Reference

This graph shows which files directly or indirectly include this file:

Classes

- class handler_event

  Handler event.

11.96 handler_event.hpp

```
00001 #ifndef TRISYCL_SYCL_HANDLER_EVENT_HPP
00002 #define TRISYCL_SYCL_HANDLER_EVENT_HPP
00003
00004 /** \file The handler event
00005
00006     Implement parallel constructions to launch kernels
00007
00008     Ronan at keryell dot FR
00009
00010     This file is distributed under the University of Illinois Open Source
00011     License. See LICENSE.TXT for details.
00012 */
00013
00014 /** \todo To be implemented */
00015 /** Handler event
00016
00017     \todo To be implemented
00018 */
00019 class handler_event {
00020 /*
00021  public:
00022   event get_kernel() const;
00023   event get_complete() const;
00024   event get_end() const;
00025 */
00026 };
```

```
00027
00028
00029 /*
00030     # Some Emacs stuff:
00031     ### Local Variables:
00032     ### ispell-local-dictionary: "american"
00033     ### eval: (flyspell-prog-mode)
00034     ### End:
00035 */
00036
00037 #endif // TRISYCL_SYCL_HANDLER_EVENT_HPP
```

11.97 include/CL/sycl/id.hpp File Reference

```
#include <algorithm>
#include <cstddef>
#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for id.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::item< Dimensions >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...
- class cl::sycl::id< Dimensions >

  Define a multi-dimensional index, used for example to locate a work item. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- auto cl::sycl::make_id (id< 1 > i)

  Implement a make_id to construct an id<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_id (id< 2 > i)
- auto cl::sycl::make_id (id< 3 > i)
- template<typename... BasicType>
  auto cl::sycl::make_id (BasicType... Args)

  Construct an id<> from a function call with arguments, like make_id(1, 2, 3)

11.98 id.hpp

```
#ifndef TRISYCL_SYCL_ID_HPP
#define TRISYCL_SYCL_ID_HPP

/** \file The OpenCL SYCL id<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <cstddef>

#include "CL/sycl/detail/small_array.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

template <int Dimensions> class item;

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** Define a multi-dimensional index, used for example to locate a work
    item
*/
template <int Dimensions = 1>
class id : public detail::small_array_123<
             std::size_t,
             id<Dimensions>,
             Dimensions > {

public:

```

```
  // Inherit from all the constructors
  using detail::small_array_123<std::size_t,
                                id<Dimensions>,
                                Dimensions>::small_array_123;


  /// Construct an id from the dimensions of a range
  id(const range<Dimensions> &range_size)
    /** Use the fact we have a constructor of a small_array from a another
        kind of small_array
     */
    : detail::small_array_123<std::size_t, id<Dimensions>, Dimensions>
      { range_size }
  {}


  /// Construct an id from an item global_id
  id(const item<Dimensions> &rhs)
    : detail::small_array_123<std::size_t, id<Dimensions>
, Dimensions>
      { rhs.get_id() }
  {}

  /// Keep other constructors
  id() = default;

};


/** Implement a make_id to construct an id<> of the right dimension with
    implicit conversion from an initializer list for example.

    Cannot use a template on the number of dimensions because the implicit
    conversion would not be tried. */
inline auto make_id(id<1> i) { return i; }
inline auto make_id(id<2> i) { return i; }
inline auto make_id(id<3> i) { return i; }


/** Construct an id<> from a function call with arguments, like
    make_id(1, 2, 3) */
template<typename... BasicType>
auto make_id(BasicType... Args) {
  // Call constructor directly to allow narrowing
  return id<sizeof...(Args)>(Args...);
}

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ID_HPP
```

11.99 include/CL/sycl/image.hpp File Reference

OpenCL SYCL image class.

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::image< Dimensions >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.99.1 Detailed Description

OpenCL SYCL image class.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file image.hpp.

11.100 image.hpp

```
#ifndef TRISYCL_SYCL_IMAGE_HPP
#define TRISYCL_SYCL_IMAGE_HPP

/** \file

    OpenCL SYCL image class

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {

/** \addtogroup data

```

```
    @{
*/

/// \todo implement image
template <int Dimensions> struct image;


/// @} End the data Doxygen group


}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_IMAGE_HPP
```

11.101 include/CL/sycl/info/param_traits.hpp File Reference

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::info::param_traits< T, Param >

  Implement a meta-function from (T, value) to T' to express the return type value of an OpenCL function of kind (T, value)

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Macros

- #define TRISYCL_INFO_PARAM_TRAITS_ANY_T(T, RETURN_TYPE)

  To declare a param_traits returning RETURN_TYPE for function of any T.
- #define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)

  To declare a param_traits returning RETURN_TYPE for function taking a VALUE of type T.

11.101.1 Macro Definition Documentation

11.101.1.1 TRISYCL_INFO_PARAM_TRAITS

```
#define TRISYCL_INFO_PARAM_TRAITS(
              VALUE,
              RETURN_TYPE )
```

Value:

```
template <>                                                    \
  struct param_traits<decltype(VALUE), VALUE> {                  \
    using return_type = RETURN_TYPE;                             \
  };
```

To declare a param_traits returning RETURN_TYPE for function taking a VALUE of type T.

Definition at line 36 of file param_traits.hpp.

11.101.1.2 TRISYCL_INFO_PARAM_TRAITS_ANY_T

```
#define TRISYCL_INFO_PARAM_TRAITS_ANY_T(
              T,
              RETURN_TYPE )
```

Value:

```
template <T Param>                                             \
  struct param_traits<T, Param> {                                \
    using return_type = RETURN_TYPE;                             \
  };
```

To declare a param_traits returning RETURN_TYPE for function of any T.

Definition at line 26 of file param_traits.hpp.

11.102 param_traits.hpp

```
#ifndef TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
#define TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP

/** \file The OpenCL SYCL param_traits

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {
namespace info {

/** Implement a meta-function from (T, value) to T' to express the return type
    value of an OpenCL function of kind (T, value)
*/
template <typename T, T Param>
struct param_traits {
  // By default no return type
};


/// To declare a param_traits returning RETURN_TYPE for function of any T
#define TRISYCL_INFO_PARAM_TRAITS_ANY_T(T, RETURN_TYPE)      \
  template <T Param>                                         \
  struct param_traits<T, Param> {                            \
    using return_type = RETURN_TYPE;                         \
  };


/** To declare a param_traits returning RETURN_TYPE for function taking a
    VALUE of type T
*/
#define TRISYCL_INFO_PARAM_TRAITS(VALUE, RETURN_TYPE)        \
  template <>                                                \
  struct param_traits<decltype(VALUE), VALUE> {              \
    using return_type = RETURN_TYPE;                         \
  };

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_INFO_PARAM_TRAITS_HPP
```

11.103 include/CL/sycl/info/platform.hpp File Reference

```
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/info/param_traits.hpp"
```

Include dependency graph for platform.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info

Enumerations

- enum cl::sycl::info::platform : unsigned int {
  cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_PROFILE), cl::sycl::info::platform::↩
  TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_VERSION), cl::sycl::info::platform::TRISYCL_SKIP_OPE↩
  NCL =(= CL_PLATFORM_NAME), cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM↩
  _VENDOR),
  cl::sycl::info::platform::TRISYCL_SKIP_OPENCL =(= CL_PLATFORM_EXTENSIONS) }

  Platform information descriptors.

11.104 platform.hpp

```
#ifndef TRISYCL_SYCL_INFO_PLATFORM_HPP
#define TRISYCL_SYCL_INFO_PLATFORM_HPP

/** \file The OpenCL SYCL platform information parameters

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/info/param_traits.hpp"

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/
namespace info {

/** Platform information descriptors

    A SYCL platform can be queried for all of the following information
    using the get_info function.

    In this implementation, the values are mapped to OpenCL values to
    avoid further remapping later when OpenCL is used
*/
enum class platform : unsigned int {
  /** Returns the profile name (as a string_class) supported by the
      implementation.

      Can be either FULL PROFILE or EMBEDDED PROFILE.
  */
  profile TRISYCL_SKIP_OPENCL(= CL_PLATFORM_PROFILE),

  /** Returns the OpenCL software driver version string in the form major
      number.minor number (as a string_class)
  */
  version TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VERSION),

  /** Returns the name of the platform (as a string_class)
  */
  name TRISYCL_SKIP_OPENCL(= CL_PLATFORM_NAME),

  /** Returns the string provided by the platform vendor (as a string_class)
  */
  vendor TRISYCL_SKIP_OPENCL(= CL_PLATFORM_VENDOR),

  /** Returns a space-separated list of extension names supported by the
```

```
      platform (as a string_class)
  */
  extensions TRISYCL_SKIP_OPENCL(= CL_PLATFORM_EXTENSIONS),

#if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
  /** Returns the resolution of the host timer in nanoseconds as used by
      clGetDeviceAndHostTimer
  */
  host_timer_resolution
    TRISYCL_SKIP_OPENCL(= CL_PLATFORM_HOST_TIMER_RESOLUTION)
#endif
};


/** Query the return type for get_info() on platform parameter type

    This defines the meta-function
    \code
    param_traits<info::platform x, string_class>::type == string_class
    \endcode

    for all x, which means that get_info() returns always a string_class
    when asked about platform info.
*/
TRISYCL_INFO_PARAM_TRAITS_ANY_T(info::platform,
string_class)

#if CL_SYCL_LANGUAGE_VERSION >= 220 && defined(CL_VERSION_2_1)
///  get_info<host_timer_resolution>() return a cl_ulong
#ifdef TRISYCL_OPENCL
TRISYCL_INFO_PARAM_TRAITS(info::platform::host_timer_resolution, cl_ulong)
#else
TRISYCL_INFO_PARAM_TRAITS(info::platform::host_timer_resolution,
                          unsigned long int)
#endif
#endif
}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_INFO_PLATFORM_HPP
```

11.105 include/CL/sycl/platform/detail/platform.hpp File Reference

```
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/platform.hpp"
```

Include dependency graph for platform.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::platform

  An abstract class representing various models of SYCL platforms. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.106 platform.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP

/** \file The OpenCL SYCL abstract platform

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/platform.hpp"

namespace cl {
namespace sycl {

class device;

namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// An abstract class representing various models of SYCL platforms
class platform {

public:

#ifdef TRISYCL_OPENCL
  /// Return the cl_platform_id of the underlying OpenCL platform
  virtual cl_platform_id get() const = 0;


  /// Return the underlying Boost.Compute platform, if any
  virtual const boost::compute::platform &
get_boost_compute() const = 0;
#endif


  /// Return true if the platform is a SYCL host platform
  virtual bool is_host() const = 0;


  /// Query the platform for OpenCL string info::platform info
  virtual string_class get_info_string(info::platform param) const
 = 0;


  /// Specify whether a specific extension is supported on the platform.
  virtual bool has_extension(const string_class &extension) const = 0;

  /** Get all the available devices for this platform

      \param[in] device_type is the device type to filter the selection
      or \c info::device_type::all by default to return all the
      devices

      \return the device list
  */
  virtual vector_class<device>
  get_devices(info::device_type device_type) const = 0;

  // Virtual to call the real destructor
  virtual ~platform() {}

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

```

```
00083 #endif // TRISYCL_SYCL_PLATFORM_DETAIL_PLATFORM_HPP
```

11.107 include/CL/sycl/platform.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/global_config.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/platform/detail/host_platform.hpp"
#include "CL/sycl/platform/detail/opencl_platform.hpp"
#include "CL/sycl/platform/detail/platform.hpp"
```

Include dependency graph for platform.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::platform

  Abstract the OpenCL platform. More...
- struct std::hash< cl::sycl::platform >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

11.108 platform.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_HPP

/** \file The OpenCL SYCL platform

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/global_config.hpp"

#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/info/device.hpp"
#include "CL/sycl/platform/detail/host_platform.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/platform/detail/opencl_platform.hpp"
#endif
#include "CL/sycl/platform/detail/platform.hpp"

namespace cl {
namespace sycl {

class device_selector;
class device;

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Abstract the OpenCL platform

    \todo triSYCL Implementation
*/
class platform
  /* Use the underlying platform implementation that can be shared in the
     SYCL model */
  : public detail::shared_ptr_implementation<platform, detail::platform> {

  // Allows the comparison operation to access the implementation
  friend shared_ptr_implementation;

public:

  // Make the implementation member directly accessible in this class
  using shared_ptr_implementation::implementation;

  /** Default constructor for platform which is the host platform

      Returns errors via the SYCL exception class.
  */
```

```
  platform() :
    shared_ptr_implementation {
detail::host_platform::instance() } {}


#ifdef TRISYCL_OPENCL
  /** Construct a platform class instance using cl_platform_id of the
      OpenCL device

      Return synchronous errors via the SYCL exception class.

      Retain a reference to the OpenCL platform.
  */
  platform(cl_platform_id platform_id)
    : platform { boost::compute::platform { platform_id } } {}


  /** Construct a platform class instance using a boost::compute::platform

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.
  */
  platform(const boost::compute::platform &p)
    : shared_ptr_implementation {
detail::opencl_platform::instance(p) } {}
#endif


  /**  Construct a platform object from the device selected by a device
       selector of the user's choice

       Returns errors via the SYCL exception class.
  */
  explicit platform(const device_selector &dev_selector) {
    detail::unimplemented();
  }


#ifdef TRISYCL_OPENCL
  /** Returns the cl_platform_id of the underlying OpenCL platform

      If the platform is not a valid OpenCL platform, for example if it is
      the SYCL host, an exception is thrown

      \todo Define a SYCL exception for this
  */
  cl_platform_id get() const {
    return implementation->get();
  }


  /** Return the underlying Boost.Compute platform if it is an
      OpenCL platform

      This is a triSYCL extension
  */
  const boost::compute::platform get_boost_compute() const {
    return implementation->get_boost_compute();
  }
#endif


  /// Get the list of all the platforms available to the application
  static vector_class<platform> get_platforms() {
    // Start with the default platform
    vector_class<platform> platforms { {} };

#ifdef TRISYCL_OPENCL
    // Then add all the OpenCL platforms
    for (const auto &d : boost::compute::system::platforms())
      platforms.emplace_back(d);
#endif

    return platforms;
  }


  /** Get the OpenCL information about the requested parameter

      \todo Add to the specification
  */
  template <typename ReturnT>
  ReturnT get_info(info::platform param) const {
    // Only strings are needed here
    return implementation->get_info_string(param);
  }
```

```
  /// Get the OpenCL information about the requested template parameter
  template <info::platform Param>
  typename info::param_traits<info::platform, Param>::return_type
  get_info() const {
    /* Forward to the implementation without using template parameter
       but with a parameter instead, since it is incompatible with
       virtual function and because fortunately only strings are
       needed here */
    return get_info<typename info::param_traits<info::platform,
                                                Param>::return_type>(Param);
  }


  /// Test if an extension is available on the platform
  bool has_extension(const string_class &extension) const {
    return implementation->has_extension(extension);
  }


  /// Test if this platform is a host platform
  bool is_host() const {
    return implementation->is_host();
  }


  /** Get all the available devices for this platform

      \param[in] device_type is the device type to filter the selection
      or \c info::device_type::all by default to return all the
      devices

      \return the device list
  */
  vector_class<device>
  get_devices(info::device_type device_type = info::device_type::all) const {
    return implementation->get_devices(device_type);
  }

};

/// @} to end the execution Doxygen group

}
}


/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::platform> {

  auto operator()(const cl::sycl::platform &p) const {
    // Forward the hashing to the implementation
    return p.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_HPP
```

11.109 include/CL/sycl/item.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/detail/linear_id.hpp"
```

```
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for item.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::item< Dimensions >

  A SYCL item stores information on a work-item with some more context such as the definition range and offset. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.

- cl::sycl

11.110 item.hpp

```
#ifndef TRISYCL_SYCL_ITEM_HPP
#define TRISYCL_SYCL_ITEM_HPP

/** \file The OpenCL SYCL item<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A SYCL item stores information on a work-item with some more context
    such as the definition range and offset.
*/
template <int Dimensions = 1>
class item {

public:

  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = Dimensions;

private:

  range<Dimensions> global_range;
  id<Dimensions> global_index;
  id<Dimensions> offset;

public:

  /** Create an item from a local size and an optional offset

      This constructor is used by the triSYCL implementation and the
      non-regression testing.
  */
  item(range<Dimensions> global_size,
       id<Dimensions> global_index,
       id<Dimensions> offset = {}) :
    global_range { global_size },
    global_index { global_index },
    offset { offset }
  {}


  /** To be able to copy and assign item, use default constructors too

      \todo Make most of them protected, reserved to implementation
  */
  item() = default;


  /** Return the constituent local or global id<> representing the
      work-item's position in the iteration space
  */
  id<Dimensions> get_id() const { return global_index; }


  /** Return the requested dimension of the constituent id<> representing
      the work-item's position in the iteration space
  */
  size_t get_id(int dimension) const { return get_id()[dimension]; }


  /** Return the constituent id<> l-value representing the work-item's
      position in the iteration space in the given dimension
  */
  auto &operator[](int dimension) { return global_index[dimension]; }


  /** Returns a range<> representing the dimensions of the range of
```

```
      possible values of the item
  */
  range<Dimensions> get_range() const { return global_range; }


  /** Returns an id<> representing the n-dimensional offset provided to
      the parallel_for and that is added by the runtime to the global-ID
      of each work-item, if this item represents a global range

      For an item representing a local range of where no offset was passed
      this will always return an id of all 0 values.
  */
  id<Dimensions> get_offset() const { return offset; }


  /** Return the linearized ID in the item's range

      Computed as the flatted ID after the offset is subtracted.
  */
  size_t get_linear_id() const {
    return detail::linear_id(get_range(), get_id(), get_offset());
  }


  /** For the implementation, need to set the global index

      \todo Move to private and add friends
  */
  void set(id<Dimensions> Index) { global_index = Index; }


  /// Display the value for debugging and validation purpose
  void display() const {
    global_range.display();
    global_index.display();
    offset.display();
  }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ITEM_HPP
```

11.111 include/CL/sycl/kernel/detail/kernel.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for kernel.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::kernel

  Abstract SYCL kernel. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_ParallelForKernel_RANGE(N)

  Launch a kernel with a range<>

11.111.1 Macro Definition Documentation

11.111.1.1 TRISYCL_ParallelForKernel_RANGE

```
#define TRISYCL_ParallelForKernel_RANGE(
              N )
```

Value:

```
virtual void parallel_for(std::shared_ptr<detail::task> task, std::shared_ptr<detail::queue> q,
          \
                            const range<N> &num_work_items) = 0;
```

Launch a kernel with a range<>

Do not use a template since it does not work with virtual functions

Todo Think to a cleaner solution

Definition at line 63 of file kernel.hpp.

11.112 kernel.hpp

```
00001 #ifndef TRISYCL_SYCL_KERNEL_DETAIL_KERNEL_HPP
00002 #define TRISYCL_SYCL_KERNEL_DETAIL_KERNEL_HPP
00003
00004 /** \file The OpenCL SYCL kernel
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #ifdef TRISYCL_OPENCL
00013 #include <boost/compute.hpp>
00014 #endif
00015
00016 #include "CL/sycl/detail/debug.hpp"
00017 #include "CL/sycl/detail/unimplemented.hpp"
00018 //#include "CL/sycl/info/kernel.hpp"
00019 #include "CL/sycl/queue/detail/queue.hpp"
00020 #include "CL/sycl/range.hpp"
00021
00022 namespace cl {
00023 namespace sycl {
00024 namespace detail {
00025
00026 /** \addtogroup execution Platforms, contexts, devices and queues
00027     @{
00028 */
00029
00030 /// Abstract SYCL kernel
```

```
class kernel : detail::debug<detail::kernel> {

 public:

#ifdef TRISYCL_OPENCL
  /** Return the OpenCL kernel object for this kernel

      Retains a reference to the returned cl_kernel object. Caller
      should release it when finished.
  */
  virtual cl_kernel get() const = 0;


  /** Return the Boost.Compute OpenCL kernel object for this kernel

      This is an extension.
  */
  virtual boost::compute::kernel &get_boost_compute() = 0;
#endif


  /// Launch a single task of the kernel
  virtual void single_task(std::shared_ptr<detail::task> task,
                           std::shared_ptr<detail::queue> q) = 0;


  /** Launch a kernel with a range<>

      Do not use a template since it does not work with virtual functions

      \todo Think to a cleaner solution
  */
#define TRISYCL_ParallelForKernel_RANGE(N)                          \
  virtual void parallel_for(std::shared_ptr<detail::task> task, std::shared_ptr<detail::queue> q,     \
                            const range<N> &num_work_items) = 0;

  TRISYCL_ParallelForKernel_RANGE(1)
  TRISYCL_ParallelForKernel_RANGE(2)
  TRISYCL_ParallelForKernel_RANGE(3)
#undef TRISYCL_ParallelForKernel_RANGE


  /// Return the context that this kernel is defined for
  //virtual context get_context() const;

  /// Return the program that this kernel is part of
  //virtual program get_program() const;

  // Virtual to call the real destructor
  virtual ~kernel() {}

};

/// @} End the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_DETAIL_KERNEL_KERNEL_HPP
```

11.113 include/CL/sycl/kernel.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
```

```
#include "CL/sycl/kernel/detail/opencl_kernel.hpp"
```

Include dependency graph for kernel.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::kernel

  SYCL kernel. More...
- struct std::hash< cl::sycl::kernel >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- std

11.114 kernel.hpp

```
#ifndef TRISYCL_SYCL_KERNEL_HPP
#define TRISYCL_SYCL_KERNEL_HPP

/** \file The OpenCL SYCL kernel

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/shared_ptr_implementation.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
//#include "CL/sycl/info/kernel.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/kernel/detail/opencl_kernel.hpp"
#endif

namespace cl {
namespace sycl {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** SYCL kernel

    \todo To be implemented

    \todo Check specification
*/
class kernel
  /* Use the underlying kernel implementation that can be shared in
     the SYCL model */
  : public detail::shared_ptr_implementation<kernel, detail::kernel> {

  // The type encapsulating the implementation
  using implementation_t = typename kernel::shared_ptr_implementation;

  // The handler class uses the implementation
  friend class handler;

  // Allows the comparison operation to access the implementation
  friend implementation_t;

 public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** The default object is not valid because there is no program or
      \code cl_kernel \endcode associated with it */
  kernel() = delete;

#ifdef TRISYCL_OPENCL
  /** Constructor for SYCL kernel class given an OpenCL kernel object
      with set arguments, valid for enqueuing

      Retains a reference to the \p cl_kernel object. The Caller
      should release the passed cl_kernel object when it is no longer
      needed.
  */
  kernel(cl_kernel k) : kernel { boost::compute::kernel { k } } {}


  /** Construct a kernel class instance using a boost::compute::kernel

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.
  */
  kernel(const boost::compute::kernel &k)
    : implementation_t { detail::opencl_kernel::instance(k) } {}


  /** Return the OpenCL kernel object for this kernel
```

```

      Retains a reference to the returned cl_kernel object. Caller
      should release it when finished.
  */
  cl_kernel get() const {
    return implementation->get();
  }
#endif


#if 0
  /// Return the context that this kernel is defined for
  //context get_context() const;

  /// Return the program that this kernel is part of
  //program get_program() const;

  /** Query information from the kernel object using the
      info::kernel_info descriptor.
  */
  template <info::kernel param>
  typename info::param_traits<info::kernel, param>::type
    get_info() const {
    detail::unimplemented();
  }
#endif

};

/// @} End the execution Doxygen group

}
}


/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::kernel> {

  auto operator()(const cl::sycl::kernel &k) const {
    // Forward the hashing to the implementation
    return k.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_KERNEL_HPP
```

11.115 include/CL/sycl/kernel/detail/opencl_kernel.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for opencl_kernel.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::opencl_kernel

  An abstraction of the OpenCL kernel.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Macros

- #define TRISYCL_ParallelForKernel_RANGE(N)

  Launch an OpenCL kernel with a range<>

11.115.1 Macro Definition Documentation

11.115.1.1 TRISYCL_ParallelForKernel_RANGE

```
#define TRISYCL_ParallelForKernel_RANGE(
              N )
```

Value:

```
void parallel_for(std::shared_ptr<detail::task> task,                 \
                  std::shared_ptr<detail::queue> q,                    \
                  const range<N> &num_work_items) override {           \
    static_assert(sizeof(range<N>::value_type) == sizeof(size_t),      \
                  "num_work_items::value_type compatible with "        \
                  "Boost.Compute");                                    \
    q->get_boost_compute().enqueue_nd_range_kernel                     \
      (k,                                                              \
       static_cast<size_t>(N),                                         \
       NULL,                                                           \
       static_cast<const size_t *>(num_work_items.data()),             \
       NULL);                                                          \
    /* For now use a crude synchronization mechanism to map directly a  \
       host task to an accelerator task */                             \
    q->get_boost_compute().finish();                                   \
  };
```

Launch an OpenCL kernel with a range<>

Do not use a template since it does not work with virtual functions

Todo Think to a cleaner solution

Todo Remove either task or q

Definition at line 108 of file opencl_kernel.hpp.

11.116 opencl_kernel.hpp

```
#ifndef TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP
#define TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP

/** \file The OpenCL SYCL kernel

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
//#include "CL/sycl/info/kernel.hpp"
#include "CL/sycl/kernel/detail/kernel.hpp"
#include "CL/sycl/queue/detail/queue.hpp"


namespace cl {
namespace sycl {
namespace detail {

/// An abstraction of the OpenCL kernel
class opencl_kernel : public detail::kernel,
                      detail::debug<opencl_kernel> {

  /// Use the Boost Compute abstraction of the OpenCL kernel
  boost::compute::kernel k;

  /** A cache to always return the same alive kernel for a given
      OpenCL kernel

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_kernel, detail::opencl_kernel>
cache;

  opencl_kernel(const boost::compute::kernel &k) : k { k } {}

 public:

  ///// Get a singleton instance of the opencl_device
  static std::shared_ptr<opencl_kernel>
  instance(const boost::compute::kernel &k) {
    return cache.get_or_register(k.get(),
                                 [&] { return new opencl_kernel { k }; });
  }

  /** Return the underlying OpenCL object

      \todo Improve the spec to deprecate C OpenCL host API and move
      to C++ instead to avoid this ugly ownership management
  */
  cl_kernel get() const override {
    /// \todo Test error and throw. Externalize this feature in Boost.Compute?
    clRetainKernel(k);
    return k.get();
  }


  /** Return the Boost.Compute OpenCL kernel object for this kernel

      This is an extension.
  */
  boost::compute::kernel &get_boost_compute() override {
    return k;
  }


  //context get_context() const override

  //program get_program() const override

#if 0
  template <info::kernel param>
  typename info::param_traits<info::kernel, param>::type
  get_info() const {
    detail::unimplemented();
  }
```

```
#endif


  /** Launch a single task of the OpenCL kernel

      \todo Remove either task or q
   */
  void single_task(std::shared_ptr<detail::task> task,
                   std::shared_ptr<detail::queue> q) override {
    q->get_boost_compute().enqueue_task(k);
    /* For now use a crude synchronization mechanism to map directly a
       host task to an accelerator task */
    q->get_boost_compute().finish();
  }


  /** Launch an OpenCL kernel with a range<>

      Do not use a template since it does not work with virtual functions

      \todo Think to a cleaner solution

      \todo Remove either task or q
  */
#define TRISYCL_ParallelForKernel_RANGE(N)                                \
  void parallel_for(std::shared_ptr<detail::task> task,                   \
                    std::shared_ptr<detail::queue> q,                     \
                    const range<N> &num_work_items) override {            \
    static_assert(sizeof(range<N>::value_type) == sizeof(size_t),         \
                  "num_work_items::value_type compatible with "           \
                  "Boost.Compute");                                       \
    q->get_boost_compute().enqueue_nd_range_kernel                        \
      (k,                                                                 \
       static_cast<size_t>(N),                                            \
       NULL,                                                              \
       static_cast<const size_t *>(num_work_items.data()),                \
       NULL);                                                             \
    /* For now use a crude synchronization mechanism to map directly a   \
       host task to an accelerator task */                                \
    q->get_boost_compute().finish();                                      \
  };

  TRISYCL_ParallelForKernel_RANGE(1)
  TRISYCL_ParallelForKernel_RANGE(2)
  TRISYCL_ParallelForKernel_RANGE(3)
#undef TRISYCL_ParallelForKernel_RANGE


  /// Unregister from the cache on destruction
  ~opencl_kernel() override {
    cache.remove(k.get());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
TRISYCL_WEAK_ATTRIB_PREFIX
detail::cache<cl_kernel, detail::opencl_kernel>
opencl_kernel::cache
TRISYCL_WEAK_ATTRIB_SUFFIX;

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_KERNEL_DETAIL_OPENCL_KERNEL_HPP
```

11.117 include/CL/sycl/math.hpp File Reference

Implement a wrapper around OpenCL math operations Joan.Thibault AT ens-rennes POINT fr This file is distributed under the University of Illinois Open Source License.

```
#include <algorithm>
#include <cmath>
```

Include dependency graph for math.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Macros

- #define TRISYCL_MATH_WRAP(FUN)
- #define TRISYCL_MATH_WRAP2(FUN)
- #define TRISYCL_MATH_WRAP2s(FUN)
- #define TRISYCL_MATH_WRAP3(FUN)
- #define TRISYCL_MATH_WRAP3s(FUN)
- #define TRISYCL_MATH_WRAP3ss(FUN)

Functions

- cl::sycl::TRISYCL_MATH_WRAP (abs) TRISYCL_MATH_WRAP(atan) TRISYCL_MATH_WRAP2s(fmax) TRISYCL_MATH_WRAP2s(fmin) TRISYCL_MATH_WRAP2s(frexp) template< typename T > T max(T x
- template<typename T >
  T cl::sycl::min (T x, T y, T z)
- cl::sycl::TRISYCL_MATH_WRAP2s (modf) TRISYCL_MATH_WRAP3s(remquo) TRISYCL_MATH_WRA↩P2(rotate) namespace native

Variables

- T cl::sycl::y
- T T cl::sycl::z

11.117.1 Detailed Description

Implement a wrapper around OpenCL math operations Joan.Thibault AT ens-rennes POINT fr This file is distributed under the University of Illinois Open Source License.

See LICENSE.TXT for details.

Definition in file math.hpp.

11.117.2 Macro Definition Documentation

11.117.2.1 TRISYCL_MATH_WRAP

```
#define TRISYCL_MATH_WRAP(
                FUN )
```

Value:

```
template<typename T>                                    \
  T FUN(T x) {                                                                  \
    return std::FUN(x);                                                         \
  }
```

Definition at line 26 of file math.hpp.

11.117.2.2 TRISYCL_MATH_WRAP2

```
#define TRISYCL_MATH_WRAP2(
                FUN )
```

Value:

```
template<typename T>                                    \
  T FUN(T x, T y) {                                                    \
    return std::FUN(x, y);                                             \
  }
```

Definition at line 30 of file math.hpp.

11.117.2.3 TRISYCL_MATH_WRAP2s

```
#define TRISYCL_MATH_WRAP2s(
                FUN )
```

Value:

```
template<typename T, typename U>                        \
  T FUN(T x, U y) {                                                    \
    return std::FUN(x, y);                                             \
  }
```

Definition at line 34 of file math.hpp.

11.117.2.4 TRISYCL_MATH_WRAP3

```
#define TRISYCL_MATH_WRAP3(
                FUN )
```

Value:

```
template<typename T>                                    \
  T FUN(T x, T y, T z) {                                               \
    return std::FUN(x, y, z);                                          \
  }
```

Definition at line 38 of file math.hpp.

11.117.2.5 TRISYCL_MATH_WRAP3s

```
#define TRISYCL_MATH_WRAP3s(
              FUN )
```

Value:

```
template<typename T, typename U>              \
  T FUN(T x, T y, U z) {                                                  \
    return std::FUN(x, y, z);                                             \
  }
```

Definition at line 42 of file math.hpp.

11.117.2.6 TRISYCL_MATH_WRAP3ss

```
#define TRISYCL_MATH_WRAP3ss(
              FUN )
```

Value:

```
template<typename T, typename U>             \
  T FUN(T x, U y, U z) {                                                  \
    return std::FUN(x, y, z);                                             \
  }
```

Definition at line 46 of file math.hpp.

11.118 math.hpp

```
#ifndef TRISYCL_SYCL_MATH_HPP
#define TRISYCL_SYCL_MATH_HPP

/** \file
    Implement a wrapper around OpenCL math operations
    Joan.Thibault AT ens-rennes POINT fr
    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <algorithm>
#include <cmath>

// Include order and configure insensitive treating of unwanted macros
#ifdef _MSC_VER
  #ifdef min
    #undef min
  #endif
  #ifdef max
    #undef max
  #endif
#endif

namespace cl {
namespace sycl {
#define TRISYCL_MATH_WRAP(FUN) template<typename T>                           \
  T FUN(T x) {                                                                \
    return std::FUN(x);                                                       \
  }
#define TRISYCL_MATH_WRAP2(FUN) template<typename T>                          \
  T FUN(T x, T y) {                                                           \
    return std::FUN(x, y);                                                    \
```

```
  }
#define TRISYCL_MATH_WRAP2s(FUN) template<typename T, typename U>                \
  T FUN(T x, U y) {                                                              \
    return std::FUN(x, y);                                                       \
  }
#define TRISYCL_MATH_WRAP3(FUN) template<typename T>                             \
  T FUN(T x, T y, T z) {                                                         \
    return std::FUN(x, y, z);                                                    \
  }
#define TRISYCL_MATH_WRAP3s(FUN) template<typename T, typename U>                \
  T FUN(T x, T y, U z) {                                                         \
    return std::FUN(x, y, z);                                                    \
  }
#define TRISYCL_MATH_WRAP3ss(FUN) template<typename T, typename U>               \
  T FUN(T x, U y, U z) {                                                         \
    return std::FUN(x, y, z);                                                    \
  }

TRISYCL_MATH_WRAP(abs)//I
//*TRISYCL_MATH_WRAP2(abs_diff)//I
//*TRISYCL_MATH_WRAP2(add_sat)//I
TRISYCL_MATH_WRAP(acos)
TRISYCL_MATH_WRAP(acosh)
//*TRISYCL_MATH_WRAP(acospi)
TRISYCL_MATH_WRAP(asin)
TRISYCL_MATH_WRAP(asinh)
//*TRISYCL_MATH_WRAP(asinpi)
TRISYCL_MATH_WRAP(atan) // atan(y/x)
TRISYCL_MATH_WRAP2(atan2)
TRISYCL_MATH_WRAP(atanh)
//*TRISYCL_MATH_WRAP(atanpi)
//*TRISYCL_MATH_WRAP2(atan2pi)
TRISYCL_MATH_WRAP(cbrt)
TRISYCL_MATH_WRAP(ceil)
//*TRISYCL_MATH_WRAP3ss(clamp)//I
//geninteger clamp(geninteger, sgeninteger, sgeninteger)
//*TRISYCL_MATH_WRAP(clz)
TRISYCL_MATH_WRAP2(copysign)
TRISYCL_MATH_WRAP(cos)
TRISYCL_MATH_WRAP(cosh)
//*TRISYCL_MATH_WRAP(cospi)
TRISYCL_MATH_WRAP(erfc)
TRISYCL_MATH_WRAP(erf)
TRISYCL_MATH_WRAP(exp)
TRISYCL_MATH_WRAP(exp2)
//*TRISYCL_MATH_WRAP(exp10)
TRISYCL_MATH_WRAP(expm1)
TRISYCL_MATH_WRAP(fabs)
TRISYCL_MATH_WRAP2(fdim)
TRISYCL_MATH_WRAP(floor)
TRISYCL_MATH_WRAP3(fma)
/* genfloat fmax ( genfloat x, genfloat y)
 * genfloat fmax ( genfloat x, sgenfloat y)
 */
TRISYCL_MATH_WRAP2s(fmax)
TRISYCL_MATH_WRAP2s(fmin)
TRISYCL_MATH_WRAP2(fmod)
//*TRISYCL_MATH_WRAP2s(fract)
TRISYCL_MATH_WRAP2s(frexp)
//*TRISYCL_MATH_WRAP(hadd)
TRISYCL_MATH_WRAP2(hypot)
//log
//ilogb
//ldexp
TRISYCL_MATH_WRAP(lgamma)
//*TRISYCL_MATH_WRAP2s(lgamma_r)
TRISYCL_MATH_WRAP(log)
TRISYCL_MATH_WRAP(log2)
TRISYCL_MATH_WRAP(log10)
TRISYCL_MATH_WRAP(log1p)
TRISYCL_MATH_WRAP(logb)
//*TRISYCL_MATH_WRAP3(mad)
//*TRISYCL_MATH_WRAP3(mad_hi)//I
//*TRISYCL_MATH_WRAP3(mad_sat)
//
//TRISYCL_MATH_WRAP3s(max) //I
template<typename T>
T max(T x, T y, T z) {
  return std::max(x, std::max(y, z));
}
/* geninteger max (geninteger, geninteger)
 * geninteger max (geninteger, sgeninteger)
 */
  // Just import std::max from <algorithm>
  using std::max;

//*TRISYCL_MATH_WRAP2(maxmag)
```

```
//
//TRISYCL_MATH_WRAP3s(min) //I
template<typename T>
T min(T x, T y, T z) {
  return std::min(x, std::min(y, z));
}
/* geninteger min (geninteger, geninteger)
 * geninteger min (geninteger, sgeninteger)
 */
  // Just import std::min from <algorithm>
  using std::min;

//*TRISYCL_MATH_WRAP2(minmag)
TRISYCL_MATH_WRAP2s(modf)
//*TRISYCL_MATH_WRAP2(mul_hi)//I
//nan
TRISYCL_MATH_WRAP2(pow)
//*TRISYCL_MATH_WRAP2s(posn)
//*TRISYCL_MATH_WRAP2(powr)
TRISYCL_MATH_WRAP2(remainder)
TRISYCL_MATH_WRAP3s(remquo)
//*TRISYCL_MATH_WRAP(rhadd)//I
TRISYCL_MATH_WRAP(rint)
//*TRISYCL_MATH_WRAP3s(rootn)
TRISYCL_MATH_WRAP2(rotate)//I
TRISYCL_MATH_WRAP(round)
//*TRISYCL_MATH_WRAP(rsqrt)
TRISYCL_MATH_WRAP(sin)
//*TRISYCL_MATH_WRAP2s(sincos)
TRISYCL_MATH_WRAP(sinh)
//*TRISYCL_MATH_WRAP(sinpi)
TRISYCL_MATH_WRAP(sqrt)
//*TRISYCL_MATH_WRAP2(sub_sat)
TRISYCL_MATH_WRAP(tan)
TRISYCL_MATH_WRAP(tanh)
//*TRISYCL_MATH_WRAP(tanpi)
TRISYCL_MATH_WRAP(tgamma)
TRISYCL_MATH_WRAP(trunc)
/* Integer concatenation
 * shortn upsample (charn hi, ucharn lo)
 * ushortn upsample (ucharn hi, ucharn lo)
 * intn upsample (shortn hi, ushortn lo)
 * uintn upsample (ushortn hi, ushortn lo)
 * longlongn upsample(intn hi, uintn lo)
 * ulonglongn upsample(uintn hi, uintn l)
 */
//*TRISYCL_MATH_WRAP(popcount)//I
//*TRISYCL_MATH_WRAP3(mad24)
//*TRISYCL_MATH_WRAP3(mul24)

//
namespace native {
TRISYCL_MATH_WRAP(cos)
//*TRISYCL_MATH_WRAP2(divide)
TRISYCL_MATH_WRAP(exp)
TRISYCL_MATH_WRAP(exp2)
//*TRISYCL_MATH_WRAP(exp10)
TRISYCL_MATH_WRAP(log)
TRISYCL_MATH_WRAP(log2)
TRISYCL_MATH_WRAP(log10)
//*TRISYCL_MATH_WRAP(powr)
//*TRISYCL_MATH_WRAP(recip)
//*TRISYCL_MATH_WRAP(rsqrt)
TRISYCL_MATH_WRAP(sin)
TRISYCL_MATH_WRAP(sqrt)
TRISYCL_MATH_WRAP(tan)
}
#undef TRISYCL_MATH_WRAP
#undef TRISYCL_MATH_WRAP2
#undef TRISYCL_MATH_WRAP2s
#undef TRISYCL_MATH_WRAP3
#undef TRISYCL_MATH_WRAP3s
#undef TRISYCL_MATH_WRAP3ss

}
}

#endif
```

11.119 include/CL/sycl/nd_item.hpp File Reference

```
#include <cstddef>
```

```
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for nd_item.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::nd_item< Dimensions >

  A SYCL nd_item stores information on a work-item within a work-group, with some more context such as the definition ranges. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.120 nd_item.hpp

```
#ifndef TRISYCL_SYCL_ND_ITEM_HPP
#define TRISYCL_SYCL_ND_ITEM_HPP

/** \file The OpenCL SYCL nd_item<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/linear_id.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A SYCL nd_item stores information on a work-item within a work-group,
    with some more context such as the definition ranges.
*/
template <int Dimensions = 1>
struct nd_item {
  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = Dimensions;

private:

  id<Dimensions> global_index;
  /* This is a cached value since it can be computed from global_index and
     ND_range */
  id<Dimensions> local_index;
  nd_range<Dimensions> ND_range;

public:

  /** Create an empty nd_item<> from an nd_range<>

      \todo This is for the triSYCL implementation which is expected to
      call set_global() and set_local() later. This should be hidden to
      the user.
  */
  nd_item(nd_range<Dimensions> ndr) : ND_range { ndr } {}


  /** Create a full nd_item

      \todo This is for validation purpose. Hide this to the programmer
      somehow
  */
  nd_item(id<Dimensions> global_index,
          nd_range<Dimensions> ndr) :
    global_index { global_index },
    // Compute the local index using the offset and the group size
    local_index
      { (global_index - ndr.get_offset())%id<Dimensions> { ndr.get_local() } },
    ND_range { ndr }
  {}

```

```
  /** To be able to copy and assign nd_item, use default constructors too

      \todo Make most of them protected, reserved to implementation
  */
  nd_item() = default;


  /** Return the constituent global id representing the work-item's
      position in the global iteration space
  */
  id<Dimensions> get_global() const { return global_index; }


  /** Return the constituent element of the global id representing the
      work-item's position in the global iteration space in the given
      dimension
  */
  size_t get_global(int dimension) const { return get_global()[dimension]; }


  /** Return the flattened id of the current work-item after subtracting
      the offset
  */
  size_t get_global_linear_id() const {
    return detail::linear_id(get_global_range(), get_global(), get_offset());
  }


  /** Return the constituent local id representing the work-item's
      position within the current work-group
  */
  id<Dimensions> get_local() const { return local_index; }


  /** Return the constituent element of the local id representing the
      work-item's position within the current work-group in the given
      dimension
  */
  size_t get_local(int dimension) const { return get_local()[dimension]; }


  /** Return the flattened id of the current work-item within the current
      work-group
   */
  size_t get_local_linear_id() const {
    return detail::linear_id(get_local_range(), get_local());
  }


  /** Return the constituent group group representing the work-group's
      position within the overall nd_range
  */
  id<Dimensions> get_group() const {
    /* Convert get_local_range() to an id<> to remove ambiguity into using
       implicit conversion either from range<> to id<> or the opposite */
    return get_global()/id<Dimensions> { get_local_range() };
  }


  /** Return the constituent element of the group id representing the
      work-group;s position within the overall nd_range in the given
      dimension.
  */
  size_t get_group(int dimension) const {
    return get_group()[dimension];
  }


  /// Return the flattened id of the current work-group
  size_t get_group_linear_id() const {
    return detail::linear_id(get_num_groups(), get_group());
  }


  /// Return the number of groups in the nd_range
  id<Dimensions> get_num_groups() const {
    return get_nd_range().get_group();
  }

  /// Return the number of groups for dimension in the nd_range
  size_t get_num_groups(int dimension) const {
     return get_num_groups()[dimension];
  }
```

```


  /// Return a range<> representing the dimensions of the nd_range<>
  range<Dimensions> get_global_range() const {
    return get_nd_range().get_global();
  }


  /// Return a range<> representing the dimensions of the current work-group
  range<Dimensions> get_local_range() const {
    return get_nd_range().get_local();
  }


  /** Return an id<> representing the n-dimensional offset provided to the
      constructor of the nd_range<> and that is added by the runtime to the
      global-ID of each work-item
  */
  id<Dimensions> get_offset() const { return get_nd_range().get_offset(); }


  /// Return the nd_range<> of the current execution
  nd_range<Dimensions> get_nd_range() const { return ND_range; }


  /** Allows projection down to an item

      \todo Add to the specification
  */
  item<Dimensions> get_item() const {
    return { get_global_range(), get_global(), get_offset() };
  }


  /** Execute a barrier with memory ordering on the local address space,
      global address space or both based on the value of flag

      The current work-item will wait at the barrier until all work-items
      in the current work-group have reached the barrier.

      In addition, the barrier performs a fence operation ensuring that all
      memory accesses in the specified address space issued before the
      barrier complete before those issued after the barrier
  */
  void barrier(access::fence_space flag =
               access::fence_space::global_and_local) const {
#if defined(_OPENMP) && !defined(TRISYCL_NO_BARRIER)
    /* Use OpenMP barrier in the implementation with 1 OpenMP thread per
       work-item of the work-group */
#pragma omp barrier
#else
    // \todo To be implemented efficiently otherwise
    detail::unimplemented();
#endif
  }


  // For the triSYCL implementation, need to set the local index
  void set_local(id<Dimensions> Index) { local_index = Index; }


  // For the triSYCL implementation, need to set the global index
  void set_global(id<Dimensions> Index) { global_index = Index; }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ND_ITEM_HPP
```

11.121 include/CL/sycl/nd_range.hpp File Reference

```
#include <cstddef>
#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for nd_range.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::nd_range< Dimensions >

  A ND-range, made by a global and local range, to specify work-group and work-item organization. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.122 nd_range.hpp

```
#ifndef TRISYCL_SYCL_ND_RANGE_HPP
#define TRISYCL_SYCL_ND_RANGE_HPP

/** \file The OpenCL SYCL nd_range<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>

#include "CL/sycl/id.hpp"
#include "CL/sycl/range.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A ND-range, made by a global and local range, to specify work-group
    and work-item organization.

    The local offset is used to translate the iteration space origin if
    needed.

    \todo add copy constructors in the specification
*/
template <int Dimensions = 1>
struct nd_range {
  /// \todo add this Boost::multi_array or STL concept to the
  /// specification?
  static constexpr auto dimensionality = Dimensions;

private:

  range<dimensionality> global_range;
  range<dimensionality> local_range;
  id<dimensionality> offset;

public:

  /** Construct a ND-range with all the details available in OpenCL

      By default use a zero offset, that is iterations start at 0
   */
  nd_range(range<Dimensions> global_size,
           range<Dimensions> local_size,
           id<Dimensions> offset = {}) :
    global_range { global_size }, local_range { local_size }, offset { offset }
  { }


  /// Get the global iteration space range
  range<Dimensions> get_global() const { return global_range; }


  /// Get the local part of the iteration space range
  range<Dimensions> get_local() const { return local_range; }


  /// Get the range of work-groups needed to run this ND-range
  auto get_group() const {
    /* This is basically global_range/local_range, round up to the
       next integer, in case the global range is not a multiple of the
       local range. Note this is a motivating example to build a range
       from a scalar with a broadcasting constructor. */
    return (global_range + local_range - range<Dimensions>{ 1 })/local_range;
  }


  /// \todo get_offset() is lacking in the specification
  id<Dimensions> get_offset() const { return offset; }


  /// Display the value for debugging and validation purpose
  void display() const {
    global_range.display();
    local_range.display();
```

```
    offset.display();
  }

};

/// @} End the parallelism Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_ND_RANGE_HPP
```

11.123 include/CL/sycl/opencl_types.hpp File Reference

triSYCL wrapper for OpenCL types

```
#include <boost/preprocessor/cat.hpp>
#include <boost/preprocessor/comparison/equal.hpp>
#include <boost/preprocessor/control/if.hpp>
#include <boost/preprocessor/facilities/empty.hpp>
#include <boost/preprocessor/list/for_each.hpp>
#include <boost/preprocessor/logical/not.hpp>
#include <boost/preprocessor/logical/or.hpp>
#include <boost/preprocessor/seq/for_each.hpp>
#include <boost/preprocessor/tuple/to_list.hpp>
#include <boost/preprocessor/tuple/elem.hpp>
#include <boost/compute/types/fundamental.hpp>
```

Include dependency graph for opencl_types.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::is_wrapper< T >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Macros

- #define TRISYCL_SCALAR_TYPES
- #define TRISYCL_TYPE_NAME(T) BOOST_PP_TUPLE_ELEM(3, 0, T)
- #define TRISYCL_TYPE_CL_NAME(T) BOOST_PP_TUPLE_ELEM(3, 1, T)
- #define TRISYCL_TYPE_ACTUAL_NAME(T) BOOST_PP_TUPLE_ELEM(3, 2, T)
- #define TRISYCL_SIZED_NAME(T, size)
- #define TRISYCL_IS_WRAPPER_TRAIT(type) template <> struct is_wrapper<type> : std::true_type {};
- #define TRISYCL_WRAPPER_CLASS_2(cl_type, boost_name, scalar_type)
- #define TRISYCL_WRAPPER_CLASS_3(cl_type, boost_name, scalar_type)
- #define TRISYCL_WRAPPER_CLASS_4(cl_type, boost_name, scalar_type)
- #define TRISYCL_BOOST_COMPUTE_NAME(scalar, size) TRISYCL_SIZED_NAME(boost::compute::TR↩ ISYCL_TYPE_NAME(scalar), size)
- #define TRISYCL_TYPEDEF_TYPE(cl_type, boost_name, scalar_type) using cl_type = boost_name##_;
- #define TRISYCL_H_DEFINE_TYPE(cl_type, boost_name, scalar_type, i)
- #define TRISYCL_DEFINE_TYPES(scalar, i)
- #define TRISYCL_DECLARE_CL_TYPES(r, data, scalar)

11.123.1 Detailed Description

triSYCL wrapper for OpenCL types

Joan DOT Thibault AT ens-rennes DOT fr a.doumoulakis AT gmail DOT com

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file opencl_types.hpp.

11.123.2 Macro Definition Documentation

11.123.2.1 TRISYCL_BOOST_COMPUTE_NAME

```
#define TRISYCL_BOOST_COMPUTE_NAME(
              scalar,
              size ) TRISYCL_SIZED_NAME(boost::compute::TRISYCL_TYPE_NAME(scalar), size)
```

Definition at line 137 of file opencl_types.hpp.

11.123.2.2 TRISYCL_DECLARE_CL_TYPES

```
#define TRISYCL_DECLARE_CL_TYPES(
              r,
              data,
              scalar )
```

Value:

```
TRISYCL_DEFINE_TYPES(scalar, 1)                                          \
  TRISYCL_DEFINE_TYPES(scalar, 2)                                          \
  TRISYCL_DEFINE_TYPES(scalar, 3)                                          \
  TRISYCL_DEFINE_TYPES(scalar, 4)                                          \
  TRISYCL_DEFINE_TYPES(scalar, 8)                                          \
  TRISYCL_DEFINE_TYPES(scalar, 16)
```

Definition at line 168 of file opencl_types.hpp.

11.123.2.3 TRISYCL_DEFINE_TYPES

```
#define TRISYCL_DEFINE_TYPES(
              scalar,
              i )
```

Value:

```
TRISYCL_H_DEFINE_TYPE(TRISYCL_SIZED_NAME(
      TRISYCL_TYPE_CL_NAME(scalar), i),    \
                        TRISYCL_BOOST_COMPUTE_NAME(scalar,
      i),                 \
                        TRISYCL_TYPE_ACTUAL_NAME(scalar),
      i)
```

Definition at line 158 of file opencl_types.hpp.

11.123.2.4 TRISYCL_H_DEFINE_TYPE

```
#define TRISYCL_H_DEFINE_TYPE(
              cl_type,
              boost_name,
              scalar_type,
              i )
```

Value:

```
BOOST_PP_IF(BOOST_PP_EQUAL(i, 1), TRISYCL_TYPEDEF_TYPE,                      \
              BOOST_PP_IF(BOOST_PP_EQUAL(i, 2), TRISYCL_WRAPPER_CLASS_2,       \
                          BOOST_PP_IF(BOOST_PP_EQUAL(i, 3),                    \
                                      TRISYCL_WRAPPER_CLASS_3,                 \
                                      BOOST_PP_IF(BOOST_PP_EQUAL(i, 4),        \
                                                  TRISYCL_WRAPPER_CLASS_4,     \
                                                  TRISYCL_TYPEDEF_TYPE))))     \
  (cl_type, boost_name, scalar_type)
```

Definition at line 147 of file opencl_types.hpp.

11.123.2.5 TRISYCL_IS_WRAPPER_TRAIT

```
#define TRISYCL_IS_WRAPPER_TRAIT(
              type ) template <> struct is_wrapper<type> :  std::true_type {};
```

Definition at line 83 of file opencl_types.hpp.

11.123.2.6 TRISYCL_SCALAR_TYPES

```
#define TRISYCL_SCALAR_TYPES
```

Value:

```
BOOST_PP_TUPLE_TO_LIST(                                                    \
    10,                                                                     \
    (                                                                       \
      ( char   ,cl_char,   char),                                           \
      ( uchar  ,cl_uchar,  unsigned char),                                  \
      ( short  ,cl_short,  short int),                                      \
      ( ushort ,cl_ushort, unsigned short int),                             \
      ( int    ,cl_int,    int),                                            \
      ( uint   ,cl_uint,   unsigned int),                                   \
      ( long   ,cl_long,   long int),                                       \
      ( ulong  ,cl_ulong,  unsigned long int),                              \
      ( float  ,cl_float,  float),                                          \
      ( double ,cl_double, double)                                          \
      )                                                                     \
    )
```

Definition at line 36 of file opencl_types.hpp.

11.123.2.7 TRISYCL_SIZED_NAME

```
#define TRISYCL_SIZED_NAME(
               T,
               size )
```

Value:

```
BOOST_PP_IF(                                                               \
    BOOST_PP_EQUAL(size, 1), T,                                             \
    BOOST_PP_CAT(T, size))
```

Definition at line 59 of file opencl_types.hpp.

11.123.2.8 TRISYCL_TYPE_ACTUAL_NAME

```
#define TRISYCL_TYPE_ACTUAL_NAME(
               T ) BOOST_PP_TUPLE_ELEM(3, 2, T)
```

Definition at line 56 of file opencl_types.hpp.

11.123.2.9 TRISYCL_TYPE_CL_NAME

```
#define TRISYCL_TYPE_CL_NAME(
               T ) BOOST_PP_TUPLE_ELEM(3, 1, T)
```

Definition at line 55 of file opencl_types.hpp.

11.123.2.10 TRISYCL_TYPE_NAME

```
#define TRISYCL_TYPE_NAME(
              T ) BOOST_PP_TUPLE_ELEM(3, 0, T)
```

Definition at line 54 of file opencl_types.hpp.

11.123.2.11 TRISYCL_TYPEDEF_TYPE

```
#define TRISYCL_TYPEDEF_TYPE(
              cl_type,
              boost_name,
              scalar_type ) using cl_type = boost_name##_;
```

Definition at line 143 of file opencl_types.hpp.

11.123.2.12 TRISYCL_WRAPPER_CLASS_2

```
#define TRISYCL_WRAPPER_CLASS_2(
              cl_type,
              boost_name,
              scalar_type )
```

Value:

```
class cl_type {                                                          \
    ::cl_type self;                                                       \
                                                                          \
  public:                                                                 \
    cl_type () = default;                                                 \
    cl_type (::cl_type self_) : self { self_ } {}                         \
    cl_type (scalar_type x, scalar_type y) : self { x, y } {}             \
    auto& x() { return self.s[0]; }                                       \
    auto& y() { return self.s[1]; }                                       \
    auto& unwrap() const { return self; }};                               \
  TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

Definition at line 91 of file opencl_types.hpp.

11.123.2.13 TRISYCL_WRAPPER_CLASS_3

```
#define TRISYCL_WRAPPER_CLASS_3(
              cl_type,
              boost_name,
              scalar_type )
```

Value:

```
class cl_type {                                                          \
    ::cl_type self;                                                       \
                                                                          \
  public:                                                                 \
    cl_type () = default;                                                 \
    cl_type (::cl_type self_) : self { self_ } {}                         \
    cl_type (scalar_type x, scalar_type y, scalar_type z) :               \
      self { x, y, z } {}                                                 \
    auto& x() { return self.s[0]; }                                       \
    auto& y() { return self.s[1]; }                                       \
    auto& z() { return self.s[2]; }                                       \
    auto& unwrap() const { return self; }};                               \
  TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

Definition at line 104 of file opencl_types.hpp.

11.123.2.14 TRISYCL_WRAPPER_CLASS_4

```
#define TRISYCL_WRAPPER_CLASS_4(
               cl_type,
               boost_name,
               scalar_type )
```

Value:

```
class cl_type {                                                                  \
    ::cl_type self;                                                               \
                                                                                  \
  public:                                                                         \
    cl_type () = default;                                                         \
    cl_type (::cl_type self_) : self { self_ } {}                                 \
    cl_type (scalar_type x, scalar_type y, scalar_type z, scalar_type w) :        \
      self { x, y, z, w } {}                                                      \
    auto& x() { return self.s[0]; }                                               \
    auto& y() { return self.s[1]; }                                               \
    auto& z() { return self.s[2]; }                                               \
    auto& w() { return self.s[3]; }                                               \
    auto& unwrap() const { return self; }};                                       \
  TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

Definition at line 119 of file opencl_types.hpp.

11.124 opencl_types.hpp

```
#ifndef TRISYCL_SYCL_OPENCL_TYPES_HPP
#define TRISYCL_SYCL_OPENCL_TYPES_HPP

/** \file
    triSYCL wrapper for OpenCL types

    Joan DOT Thibault AT ens-rennes DOT fr
    a.doumoulakis AT gmail DOT com

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <boost/preprocessor/cat.hpp>
#include <boost/preprocessor/comparison/equal.hpp>
#include <boost/preprocessor/control/if.hpp>
#include <boost/preprocessor/facilities/empty.hpp>
#include <boost/preprocessor/list/for_each.hpp>
#include <boost/preprocessor/logical/not.hpp>
#include <boost/preprocessor/logical/or.hpp>
#include <boost/preprocessor/seq/for_each.hpp>
#include <boost/preprocessor/tuple/to_list.hpp>
#include <boost/preprocessor/tuple/elem.hpp>

#ifdef TRISYCL_OPENCL
#include <boost/compute/types/fundamental.hpp>
#else
#include "CL/sycl/vec.hpp"
#endif


namespace cl {
namespace sycl {

// List of the cl_types that will be iterated upon to generate the vector types
#define TRISYCL_SCALAR_TYPES                                                      \
  BOOST_PP_TUPLE_TO_LIST(                                                        \
    10,                                                                          \
    (                                                                            \
      ( char   ,cl_char,   char),                                                \
      ( uchar  ,cl_uchar,  unsigned char),                                       \
      ( short  ,cl_short,  short int),                                           \
      ( ushort ,cl_ushort, unsigned short int),                                  \
      ( int    ,cl_int,    int),                                                 \
      ( uint   ,cl_uint,   unsigned int),                                        \
      ( long   ,cl_long,   long int),                                            \
```

```
      ( ulong  ,cl_ulong,  unsigned long int),                                  \
      ( float  ,cl_float,  float),                                             \
      ( double ,cl_double, double)                                             \
      )                                                                        \
    )

// Accessors to get the type name, the cl_type name and the actual type
#define TRISYCL_TYPE_NAME(T)         BOOST_PP_TUPLE_ELEM(3, 0, T)
#define TRISYCL_TYPE_CL_NAME(T)      BOOST_PP_TUPLE_ELEM(3, 1, T)
#define TRISYCL_TYPE_ACTUAL_NAME(T)  BOOST_PP_TUPLE_ELEM(3, 2, T)

// Return the name of the type concatenated with the size unless the size is 1
#define TRISYCL_SIZED_NAME(T, size)                                            \
  BOOST_PP_IF(                                                                 \
    BOOST_PP_EQUAL(size, 1), T,                                                \
    BOOST_PP_CAT(T, size))

#ifndef TRISYCL_OPENCL

/* If we are not using Boost Compute we just every cl_type with its \c vec
   equivalent
*/
#define TRISYCL_DEFINE_TYPES(scalar, i)                                        \
  using TRISYCL_SIZED_NAME(TRISYCL_TYPE_CL_NAME(scalar), i) =                  \
    BOOST_PP_CAT(TRISYCL_TYPE_NAME(scalar), i);

#else

/* When passing the arguments to Boost Compute, we need to know if we have to
   unwrap our type and give Boost Compute the actual OpenCL type. To do this
   we define this meta-function that returns true if \c T is a wrapper type
   to an OpenCL type. For example \c is_wrapper<cl::sycl::int3>::value == true
*/
template<class T>
struct is_wrapper : std::false_type {};

#define TRISYCL_IS_WRAPPER_TRAIT(type)                                         \
template <> struct is_wrapper<type> : std::true_type {};

/* We define 3 different wrapper classes around OpenCL types.
   These classes allow us to use t.x() and t.y() for vector types of size 2.
   And in addition we can also use t.z() and t.w() for vector types of size
   3 and 4 similarly to OpenCL vector types
*/
#define TRISYCL_WRAPPER_CLASS_2(cl_type, boost_name, scalar_type)              \
  class cl_type {                                                              \
    ::cl_type self;                                                            \
                                                                               \
  public:                                                                      \
    cl_type () = default;                                                      \
    cl_type (::cl_type self_) : self { self_ } {}                              \
    cl_type (scalar_type x, scalar_type y) : self { x, y } {}                  \
    auto& x() { return self.s[0]; }                                            \
    auto& y() { return self.s[1]; }                                            \
    auto& unwrap() const { return self; }};                                    \
  TRISYCL_IS_WRAPPER_TRAIT(cl_type)

#define TRISYCL_WRAPPER_CLASS_3(cl_type, boost_name, scalar_type)              \
  class cl_type {                                                              \
    ::cl_type self;                                                            \
                                                                               \
  public:                                                                      \
    cl_type () = default;                                                      \
    cl_type (::cl_type self_) : self { self_ } {}                              \
    cl_type (scalar_type x, scalar_type y, scalar_type z) :                    \
      self { x, y, z } {}                                                      \
    auto& x() { return self.s[0]; }                                            \
    auto& y() { return self.s[1]; }                                            \
    auto& z() { return self.s[2]; }                                            \
    auto& unwrap() const { return self; }};                                    \
  TRISYCL_IS_WRAPPER_TRAIT(cl_type)

#define TRISYCL_WRAPPER_CLASS_4(cl_type, boost_name, scalar_type)              \
  class cl_type {                                                              \
    ::cl_type self;                                                            \
                                                                               \
  public:                                                                      \
    cl_type () = default;                                                      \
    cl_type (::cl_type self_) : self { self_ } {}                              \
    cl_type (scalar_type x, scalar_type y, scalar_type z, scalar_type w) :     \
      self { x, y, z, w } {}                                                   \
    auto& x() { return self.s[0]; }                                            \
    auto& y() { return self.s[1]; }                                            \
    auto& z() { return self.s[2]; }                                            \
    auto& w() { return self.s[3]; }                                            \
    auto& unwrap() const { return self; }};                                    \
  TRISYCL_IS_WRAPPER_TRAIT(cl_type)
```

```


// Return the Boost Compute type for OpenCL vector types
#define TRISYCL_BOOST_COMPUTE_NAME(scalar, size)                                \
  TRISYCL_SIZED_NAME(boost::compute::TRISYCL_TYPE_NAME(scalar), size)

/* For vector types of size above 4 and 1 we typedef the vector type with
   its Boost Compute equivalent
*/
#define TRISYCL_TYPEDEF_TYPE(cl_type, boost_name, scalar_type)                   \
  using cl_type = boost_name##_;

// Helper macro to properly define each OpenCL type
#define TRISYCL_H_DEFINE_TYPE(cl_type, boost_name, scalar_type, i)              \
  BOOST_PP_IF(BOOST_PP_EQUAL(i, 1), TRISYCL_TYPEDEF_TYPE,                       \
              BOOST_PP_IF(BOOST_PP_EQUAL(i, 2), TRISYCL_WRAPPER_CLASS_2,        \
                          BOOST_PP_IF(BOOST_PP_EQUAL(i, 3),                     \
                                      TRISYCL_WRAPPER_CLASS_3,                  \
                                      BOOST_PP_IF(BOOST_PP_EQUAL(i, 4),         \
                                                  TRISYCL_WRAPPER_CLASS_4,      \
                                                  TRISYCL_TYPEDEF_TYPE))))      \
   (cl_type, boost_name, scalar_type)

// Helper macro to properly define each OpenCL type
#define TRISYCL_DEFINE_TYPES(scalar, i)                                         \
  TRISYCL_H_DEFINE_TYPE(TRISYCL_SIZED_NAME(TRISYCL_TYPE_CL_NAME(scalar), i),    \
                        TRISYCL_BOOST_COMPUTE_NAME(scalar, i),                  \
                        TRISYCL_TYPE_ACTUAL_NAME(scalar), i)

#endif

/* We declare vector types of sizes 1,2,3,4,8 and 16 as per
   the SYCL specification
*/
#define TRISYCL_DECLARE_CL_TYPES(r, data, scalar)                               \
  TRISYCL_DEFINE_TYPES(scalar, 1)                                               \
  TRISYCL_DEFINE_TYPES(scalar, 2)                                               \
  TRISYCL_DEFINE_TYPES(scalar, 3)                                               \
  TRISYCL_DEFINE_TYPES(scalar, 4)                                               \
  TRISYCL_DEFINE_TYPES(scalar, 8)                                               \
  TRISYCL_DEFINE_TYPES(scalar, 16)

// Generate the vector types for all listed scalar types
BOOST_PP_LIST_FOR_EACH(TRISYCL_DECLARE_CL_TYPES, _,
TRISYCL_SCALAR_TYPES)

} // sycl
} // cl


// Undef macros to avoid name collision
#undef TRISYCL_SCALAR_TYPES
#undef TRISYCL_TYPE_NAME
#undef TRISYCL_TYPE_CL_NAME
#undef TRISYCL_TYPE_ACTUAL_NAME
#undef TRISYCL_SIZED_NAME
#undef TRISYCL_IS_WRAPPER_TRAIT
#undef TRISYCL_WRAPPER_CLASS_2
#undef TRISYCL_WRAPPER_CLASS_3
#undef TRISYCL_WRAPPER_CLASS_4
#undef TRISYCL_BOOST_COMPUTE_NAME
#undef TRISYCL_TYPEDEF_TYPE
#undef TRISYCL_DEFINE_TYPES
#undef TRISYCL_DECLARE_CL_TYPES

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_OPENCL_TYPES_HPP
```

11.125 include/CL/sycl/parallelism/detail/parallelism.hpp File Reference

Implement the detail of the parallel constructions to launch kernels.

```
#include <cstddef>
#include <boost/multi_array.hpp>
```

```
#include "CL/sycl/group.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"
```

Include dependency graph for parallelism.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::parallel_for_iterate< level, Range, ParallelForFunctor, Id >

  A recursive multi-dimensional iterator that ends up calling f. More...
- struct cl::sycl::detail::parallel_OpenMP_for_iterate< level, Range, ParallelForFunctor, Id >

  A top-level recursive multi-dimensional iterator variant using OpenMP. More...
- struct cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >

  Stop the recursion when level reaches 0 by simply calling the kernel functor with the constructed id. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

Functions

- template<int Dimensions = 1, typename ParallelForFunctor , typename Id >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, Id)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f, item< Dimensions >)

  Implementation of a data parallel computation with parallelism specified at launch time by a range<>.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (range< Dimensions > r, ParallelForFunctor f)

  Calls the appropriate ternary parallel_for overload based on the index type of the kernel function object f.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_global_offset (range< Dimensions > global_size, id< Dimensions > offset, ParallelForFunctor f)

  Implementation of parallel_for with a range<> and an offset.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement a variation of parallel_for to take into account a nd_range<>
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workgroup (nd_range< Dimensions > r, ParallelForFunctor f)

  Implement the loop on the work-groups.
- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::detail::parallel_for_workitem (const group< Dimensions > &g, ParallelForFunctor f)

  Implement the loop on the work-items inside a work-group.

11.125.1 Detailed Description

Implement the detail of the parallel constructions to launch kernels.

Ronan at keryell dot FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file parallelism.hpp.

11.126 parallelism.hpp

```
#ifndef TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
#define TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP

/** \file

    Implement the detail of the parallel constructions to launch kernels

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <boost/multi_array.hpp>

#include "CL/sycl/group.hpp"
#include "CL/sycl/id.hpp"
#include "CL/sycl/item.hpp"
#include "CL/sycl/nd_item.hpp"
#include "CL/sycl/nd_range.hpp"
#include "CL/sycl/range.hpp"

#ifdef _OPENMP
#include <omp.h>
#endif


/** \addtogroup parallelism
    @{
*/

namespace cl {
namespace sycl {
namespace detail {


/** A recursive multi-dimensional iterator that ends up calling f

    The iteration order may be changed later.

    Since partial specialization of function template is not possible in
    C++14, use a class template instead with everything in the
    constructor.
*/
template <std::size_t level,
          typename Range,
          typename ParallelForFunctor,
          typename Id>
struct parallel_for_iterate {
  parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
    for (boost::multi_array_types::index _sycl_index = 0,
           _sycl_end = r[Range::dimensionality - level];
         _sycl_index < _sycl_end;
         _sycl_index++) {
      // Set the current value of the index for this dimension
      index[Range::dimensionality - level] = _sycl_index;
      // Iterate further on lower dimensions
      parallel_for_iterate<level - 1,
                           Range,
                           ParallelForFunctor,
                           Id> { r, f, index };
    }
  }
};


/** A top-level recursive multi-dimensional iterator variant using OpenMP

    Only the top-level loop uses OpenMP and goes on with the normal
    recursive multi-dimensional.
*/
template <std::size_t level,
          typename Range,
          typename ParallelForFunctor,
          typename Id>
struct parallel_OpenMP_for_iterate {
  parallel_OpenMP_for_iterate(Range r, ParallelForFunctor &f) {
    // Create the OpenMP threads before the for-loop to avoid creating an
    // index in each iteration
#pragma omp parallel
    {
      // Allocate an OpenMP thread-local index
      Id index;
```

```
      // Make a simple loop end condition for OpenMP
      boost::multi_array_types::index _sycl_end =
        r[Range::dimensionality - level];
      /* Distribute the iterations on the OpenMP threads. Some OpenMP
         "collapse" could be useful for small iteration space, but it
         would need some template specialization to have real contiguous
         loop nests */
#pragma omp for
      for (boost::multi_array_types::index _sycl_index = 0;
           _sycl_index < _sycl_end;
           _sycl_index++) {
        // Set the current value of the index for this dimension
        index[Range::dimensionality - level] = _sycl_index;
        // Iterate further on lower dimensions
        parallel_for_iterate<level - 1,
                             Range,
                             ParallelForFunctor,
                             Id> { r, f, index };
      }
    }
  }
};


/** Stop the recursion when level reaches 0 by simply calling the
    kernel functor with the constructed id */
template <typename Range, typename ParallelForFunctor, typename Id>
struct parallel_for_iterate<0, Range, ParallelForFunctor, Id> {
  parallel_for_iterate(Range r, ParallelForFunctor &f, Id &index) {
    f(index);
  }
};


/** Implementation of a data parallel computation with parallelism
    specified at launch time by a range<>. Kernel index is id or int.

    This implementation use OpenMP 3 if compiled with the right flag.
*/
template <int Dimensions = 1, typename ParallelForFunctor, typename Id>
void parallel_for(range<Dimensions> r,
                  ParallelForFunctor f,
                  Id) {
#ifdef _OPENMP
  // Use OpenMP for the top loop level
  parallel_OpenMP_for_iterate<Dimensions,
                              range<Dimensions>,
                              ParallelForFunctor,
                              id<Dimensions>> { r, f };
#else
  // In a sequential execution there is only one index processed at a time
  id<Dimensions> index;
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       ParallelForFunctor,
                       id<Dimensions>> { r, f, index };
#endif
}


/** Implementation of a data parallel computation with parallelism
    specified at launch time by a range<>. Kernel index is item.

    This implementation use OpenMP 3 if compiled with the right flag.
*/
template <int Dimensions = 1, typename ParallelForFunctor>
void parallel_for(range<Dimensions> r,
                  ParallelForFunctor f,
                  item<Dimensions>) {
  auto reconstruct_item = [&] (id<Dimensions> l) {
    // Reconstruct the global item
    item<Dimensions> index { r, l };
    // Call the user kernel with the item<> instead of the id<>
    f(index);
  };
#ifdef _OPENMP
  // Use OpenMP for the top loop level
  parallel_OpenMP_for_iterate<Dimensions,
                              range<Dimensions>,
                              decltype(reconstruct_item),
                              id<Dimensions>> { r, reconstruct_item };
#else
  // In a sequential execution there is only one index processed at a time
  id<Dimensions> index;
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(reconstruct_item),
```

```
                        id<Dimensions>> { r, reconstruct_item, index };
#endif
}


/** Calls the appropriate ternary parallel_for overload based on the
    index type of the kernel function object f

*/
template <int Dimensions = 1, typename ParallelForFunctor>
void parallel_for(range<Dimensions> r, ParallelForFunctor f) {
  using mf_t  = decltype(std::mem_fn(&ParallelForFunctor::operator()));
  using arg_t = typename mf_t::second_argument_type;
  parallel_for(r,f,arg_t{});
}


/** Implementation of parallel_for with a range<> and an offset */
template <int Dimensions = 1, typename ParallelForFunctor>
void parallel_for_global_offset(range<Dimensions> global_size,
                                id<Dimensions> offset,
                                ParallelForFunctor f) {
  // Reconstruct the item from its id<> and its offset
  auto reconstruct_item = [&] (id<Dimensions> l) {
    // Reconstruct the global item
    item<Dimensions> index { global_size, l + offset, offset };
    // Call the user kernel with the item<> instead of the id<>
    f(index);
  };

  // First iterate on all the work-groups
  parallel_for(global_size, reconstruct_item);
}


/** Implement a variation of parallel_for to take into account a
    nd_range<>

    \todo Add an OpenMP implementation

    \todo Deal with incomplete work-groups

    \todo Implement with parallel_for_workgroup()/parallel_for_workitem()
*/
template <int Dimensions = 1, typename ParallelForFunctor>
void parallel_for(nd_range<Dimensions> r,
                  ParallelForFunctor f) {
  // To iterate on the work-group
  id<Dimensions> group;
  range<Dimensions> group_range = r.get_group();

#ifdef _OPENMP

  auto iterate_in_work_group = [&] (id<Dimensions> g) {
    //group.display();

    // Then iterate on the local work-groups
    cl::sycl::group<Dimensions> wg {g, r};
    parallel_for_workitem<Dimensions,
                          decltype(f)>(wg, f);
  };

#else

  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { r };

  // To iterate on the local work-item
  id<Dimensions> local;
  range<Dimensions> local_range = r.get_local();

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // Upgrade local_range to an id<> so that we can * with the group (an id<>)
    index.set_global(local + id<Dimensions>(local_range)*group);
    // Call the user kernel at last
    f(index);
  };

  /* To recycle the parallel_for on range<>, wrap the ParallelForFunctor f
     into another functor that iterates inside the work-group and then
     calls f */
  auto iterate_in_work_group = [&] (id<Dimensions> g) {
    //group.display();
```

```
    // Then iterate on the local work-groups
    parallel_for_iterate<Dimensions,
                         range<Dimensions>,
                         decltype(reconstruct_item),
                         id<Dimensions>> { local_range,
                                           reconstruct_item,
                                           local };
  };

#endif

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(iterate_in_work_group),
                       id<Dimensions>> { group_range,
                                         iterate_in_work_group,
                                         group };
}


/// Implement the loop on the work-groups
template <int Dimensions = 1, typename ParallelForFunctor>
void parallel_for_workgroup(nd_range<Dimensions> r,
                            ParallelForFunctor f) {
  // In a sequential execution there is only one index processed at a time
  group<Dimensions> g { r };

  // First iterate on all the work-groups
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       ParallelForFunctor,
                       group<Dimensions>> {
    r.get_group(),
    f,
    g };
}


/** Implement the loop on the work-items inside a work-group

    \todo Better type the functor
*/
template <int Dimensions, typename ParallelForFunctor>
void parallel_for_workitem(const group<Dimensions> &g,
                           ParallelForFunctor f) {
#if defined(_OPENMP) && (!defined(TRISYCL_NO_BARRIER) && !defined(_MSC_VER))
  /* To implement barriers With OpenMP, one thread is created for each
     work-item in the group and thus an OpenMP barrier has the same effect
     of an OpenCL barrier executed by the work-items in a workgroup

     The issue is that the parallel_for_workitem() execution is slow even
     when nd_item::barrier() is not used

     \todo Simplify by just using omp parallel for collapse
  */

  range<Dimensions> l_r = g.get_nd_range().get_local();
  auto tot = l_r.get(0);
  for (int i = 1; i < (int) Dimensions; ++i){
    tot *= l_r.get(i);
  }
#pragma omp parallel num_threads(tot)
  {
    nd_item<Dimensions> index { g.get_nd_range() };
    id<Dimensions> local; // to initialize correctly
#pragma omp for nowait
    for (std::size_t th_id = 0; th_id < tot; ++th_id) {
      if (Dimensions == 1) {
        local[0] = th_id;
      } else if (Dimensions == 2) {
        local[0] = th_id / l_r.get(1);
        local[1] = th_id % l_r.get(1);
      } else if (Dimensions == 3) {
        local[0] = th_id / (l_r.get(1)*l_r.get(2));
        local[1] = (th_id / l_r.get(2)) % l_r.get(1);
        local[2] = th_id % l_r.get(2);
      }
      index.set_local(local);
      index.set_global(local + id<Dimensions>(l_r)*g.get_id());
      f(index);
    }
  }
#else
  // In a sequential execution there is only one index processed at a time
  nd_item<Dimensions> index { g.get_nd_range() };
  // To iterate on the local work-item
```

```
  id<Dimensions> local;

  // Reconstruct the nd_item from its group and local id
  auto reconstruct_item = [&] (id<Dimensions> l) {
    //local.display();
    //l.display();
    // Reconstruct the global nd_item
    index.set_local(local);
    // \todo Some strength reduction here
    index.set_global(local + id<Dimensions>(g.get_local_range())*g.
get_id());
    // Call the user kernel at last
    f(index);
  };

  // Then iterate on all the work-items of the work-group
  parallel_for_iterate<Dimensions,
                       range<Dimensions>,
                       decltype(reconstruct_item),
                       id<Dimensions>> {
    g.get_local_range(),
    reconstruct_item,
    local };
#endif
}
/// @} End the parallelism Doxygen group

} // namespace detail
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PARALLELISM_DETAIL_PARALLELISM_HPP
```

11.127 include/CL/sycl/parallelism.hpp File Reference

Implement parallel constructions to launch kernels.

```
#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/id.hpp"
```

Include dependency graph for parallelism.hpp:

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- template<int Dimensions = 1, typename ParallelForFunctor >
  void cl::sycl::parallel_for_work_item (const group< Dimensions > &g, ParallelForFunctor f)

  SYCL parallel_for version that allows a Program object to be specified.

11.127.1 Detailed Description

Implement parallel constructions to launch kernels.

Ronan at keryell dot FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file parallelism.hpp.

11.128 parallelism.hpp

```
#ifndef TRISYCL_SYCL_PARALLELISM_HPP
#define TRISYCL_SYCL_PARALLELISM_HPP

/** \file

    Implement parallel constructions to launch kernels

    Ronan at keryell dot FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/parallelism/detail/parallelism.hpp"
#include "CL/sycl/id.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism
    @{
*/

/// SYCL parallel_for version that allows a Program object to be specified
/// \todo To be implemented
/* template <typename Range, typename Program, typename ParallelForFunctor>
void parallel_for(Range r, Program p, ParallelForFunctor f) {
  /// \todo deal with Program
  parallel_for(r, f);
}
*/

  /** Loop on the work-items inside a work-group

      \todo Deprecate this function in the specification to use
      instead the group method
  */
  template <int Dimensions = 1, typename ParallelForFunctor>
  void parallel_for_work_item(const group<Dimensions> &g,
                              ParallelForFunctor f) {
    g.parallel_for_work_item(f);
  }



}
}

/// @} End the parallelism Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PARALLELISM_HPP
```

11.129 include/CL/sycl/pipe/detail/pipe.hpp File Reference

```
#include <condition_variable>
#include <cstddef>
#include <mutex>
#include <deque>
#include <boost/circular_buffer.hpp>
```

Include dependency graph for pipe.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::reserve_id< T >

  A private description of a reservation station. *More...*
- class cl::sycl::detail::pipe< T >

  Implement a pipe object. *More...*

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.130 pipe.hpp

```
#ifndef TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP
#define TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP

/** \file The OpenCL SYCL pipe<> details

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <condition_variable>
#include <cstddef>
#include <mutex>
#include <deque>

#ifdef TRISYCL_MAKE_BOOST_CIRCULARBUFFER_THREAD_SAFE
/* The debug mode of boost/circular_buffer.hpp has a nasty side effect
   in multithread applications using several iterators at the same
   time even in read-only mode because the library tracks them for
   debugging purpose in a... non-thread safe way

   This is described in https://svn.boost.org/trac/boost/ticket/6277
   and fixed with https://github.com/boostorg/circular_buffer/pull/9
*/
#define BOOST_CB_DISABLE_DEBUG
#endif
#include <boost/circular_buffer.hpp>

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/// A private description of a reservation station
template <typename T>
struct reserve_id {
  /// Start of the reservation in the pipe storage
  typename boost::circular_buffer<T>::iterator start;

  /// Number of elements in the reservation
  std::size_t size;

  /* True when the reservation has been committed and is ready to be
     released */
  bool ready = false;

  /** Track a reservation not committed yet

      \param[in] start point to the start of the reservation in the
      pipe storage

      \param[in] size is the number of elements in the reservation
  */
  reserve_id(typename boost::circular_buffer<T>::iterator start,
             std::size_t size) : start { start }, size { size } {}

};


/** Implement a pipe object

    Use some mutable members so that the pipe object can be changed even
    when the accessors are captured in a lambda.
*/
template <typename T>
class pipe : public detail::debug<pipe<T>> {

public:

  using value_type = T;

  /// Implement the pipe with a circular buffer
  using implementation_t = boost::circular_buffer<value_type>;

private:

  /// The circular buffer to store the elements
  boost::circular_buffer<value_type> cb;

  /** To protect the access to the circular buffer.
```

```

      In case the object is capture in a lambda per copy, make it
      mutable. */
  mutable std::mutex cb_mutex;

  /// The queue of pending write reservations
  std::deque<reserve_id<value_type>> w_rid_q;

public:
#ifndef _MSC_VER
  using rid_iterator = typename decltype(w_rid_q)::iterator;
#else
  using rid_iterator = typename std::deque<reserve_id<value_type>>::iterator;
#endif

private:

  /// The queue of pending read reservations
  std::deque<reserve_id<value_type>> r_rid_q;

  /// Track the number of frozen elements related to read reservations
  std::size_t read_reserved_frozen;

  /// To signal that a read has been successful
  std::condition_variable read_done;

  /// To signal that a write has been successful
  std::condition_variable write_done;

  /// To control the debug mode, disabled by default
  bool debug_mode = false;

public:

  /// True when the pipe is currently used for reading
  bool used_for_reading = false;

  /// True when the pipe is currently used for writing
  bool used_for_writing = false;

  /// Create a pipe as a circular buffer of the required capacity
  pipe(std::size_t capacity) : cb { capacity }, read_reserved_frozen { 0 } { }


  /** Return the maximum number of elements that can fit in the pipe
   */
  std::size_t capacity() const {
    // No lock required since it is fixed and set at construction time
    return cb.capacity();
  }

private:

  /** Get the current number of elements in the pipe that can be read

      This is obviously a volatile value which is constrained by the
      theory of restricted relativity.

      Note that on some devices it may be costly to implement (for
      example on FPGA).
   */
  std::size_t size() const {
    TRISYCL_DUMP_T("size() cb.size() = " << cb.size()
                   << " cb.end() = " << (void *)&*cb.end()
                   << " reserved_for_reading() = " << reserved_for_reading()
                   << " reserved_for_writing() = " << reserved_for_writing());
    /* The actual number of available elements depends from the
       elements blocked by some reservations.
       This prevents a consumer to read into reserved area. */
    return cb.size() - reserved_for_reading() - reserved_for_writing();
  }


  /** Test if the pipe is empty

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      write side (for example on FPGA).
  */
  bool empty() const {
    TRISYCL_DUMP_T("empty() cb.size() = " << cb.size()
                   << " size() = " << size());
    // It is empty when the size is zero, taking into account reservations
    return size() ==  0;
  }
```

```


  /** Test if the pipe is full

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      read side (for example on FPGA).
  */
  bool full() const {
    return cb.full();
  }


public:

  /// The size() method used outside needs to lock the datastructure
  std::size_t size_with_lock() const {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return size();
  }


  /// The empty() method used outside needs to lock the datastructure
  bool empty_with_lock() const {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return empty();
  }


  // The full() method used outside needs to lock the datastructure
  bool full_with_lock() const {
    std::lock_guard<std::mutex> lg { cb_mutex };
    return full();
  }


  /** Try to write a value to the pipe

      \param[in] value is what we want to write

      \param[in] blocking specify if the call wait for the operation
      to succeed

      \return true on success

      \todo provide a && version
  */
  bool write(const T &value, bool blocking = false) {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };
    TRISYCL_DUMP_T("Write pipe full = " << full()
                   << " value = " << value);

    if (blocking)
      /* If in blocking mode, wait for the not full condition, that
         may be changed when a read is done */
      read_done.wait(ul, [&] { return !full(); });
    else if (full())
      return false;

    cb.push_back(value);
    TRISYCL_DUMP_T("Write pipe front = " << cb.front()
                   << " back = " << cb.back()
                   << " cb.begin() = " << (void *)&*cb.begin()
                   << " cb.size() = " << cb.size()
                   << " cb.end() = " << (void *)&*cb.end()
                   << " reserved_for_reading() = " << reserved_for_reading()
                   << " reserved_for_writing() = " << reserved_for_writing());
    // Notify the clients waiting to read something from the pipe
    write_done.notify_all();
    return true;
  }


  /** Try to read a value from the pipe

      \param[out] value is the reference to where to store what is
      read

      \param[in] blocking specify if the call wait for the operation
      to succeed

      \return true on success
  */
  bool read(T &value, bool blocking = false) {
```

```
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };
    TRISYCL_DUMP_T("Read pipe empty = " << empty());

    if (blocking)
      /* If in blocking mode, wait for the not empty condition, that
         may be changed when a write is done */
      write_done.wait(ul, [&] { return !empty(); });
    else if (empty())
      return false;

    TRISYCL_DUMP_T("Read pipe front = " << cb.front()
                   << " back = " << cb.back()
                   << " reserved_for_reading() = " << reserved_for_reading());
    if (read_reserved_frozen)
      /** If there is a pending reservation, read the next element to
          be read and update the number of reserved elements */
      value = cb.begin()[read_reserved_frozen++];
    else {
      /* There is no pending read reservation, so pop the read value
         from the pipe */
      value = cb.front();
      cb.pop_front();
    }

    TRISYCL_DUMP_T("Read pipe value = " << value);
    // Notify the clients waiting for some room to write in the pipe
    read_done.notify_all();
    return true;
  }


  /** Compute the amount of elements blocked by read reservations, not yet
      committed

      This includes some normal reads to pipes between/after
      un-committed reservations

      This function assumes that the data structure is locked
  */
  std::size_t reserved_for_reading() const {
    return read_reserved_frozen;
  }


  /** Compute the amount of elements blocked by write reservations, not yet
      committed

      This includes some normal writes to pipes between/after
      un-committed reservations

      This function assumes that the data structure is locked
  */
  std::size_t reserved_for_writing() const {
    if (w_rid_q.empty())
      // No on-going reservation
      return 0;
    else
      /* The reserved size is from the first element of the first
         on-going reservation up to the end of the pipe content */
      return cb.end() - w_rid_q.front().start;
  }


  /** Reserve some part of the pipe for reading

      \param[in] s is the number of element to reserve

      \param[out] rid is an iterator to a description of the
      reservation that has been done if successful

      \param[in] blocking specify if the call wait for the operation
      to succeed

      \return true if the reservation was successful
  */
  bool reserve_read(std::size_t s,
                    rid_iterator &rid,
                    bool blocking = false)  {
    // Lock the pipe to avoid being disturbed
    std::unique_lock<std::mutex> ul { cb_mutex };

    TRISYCL_DUMP_T("Before read reservation cb.size() = " << cb.size()
                   << " size() = " << size());
    if (s == 0)
      // Empty reservation requested, so nothing to do
      return false;
```

```

  if (blocking)
    /* If in blocking mode, wait for enough elements to read in the
       pipe for the reservation. This condition can change when a
       write is done */
    write_done.wait(ul, [&] { return s <= size(); });
  else if (s > size())
    // Not enough elements to read in the pipe for the reservation
    return false;

  // Compute the location of the first element of the reservation
  auto first = cb.begin() + read_reserved_frozen;
  // Increment the number of frozen elements
  read_reserved_frozen += s;
  /* Add a description of the reservation at the end of the
     reservation queue */
  r_rid_q.emplace_back(first, s);
  // Return the iterator to the last reservation descriptor
  rid = r_rid_q.end() - 1;
  TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                 << " size() = " << size());
  return true;
}


/** Reserve some part of the pipe for writing

    \param[in] s is the number of element to reserve

    \param[out] rid is an iterator to a description of the
    reservation that has been done if successful

    \param[in] blocking specify if the call wait for the operation
    to succeed

    \return true if the reservation was successful
*/
bool reserve_write(std::size_t s,
                   rid_iterator &rid,
                   bool blocking = false)  {
  // Lock the pipe to avoid being disturbed
  std::unique_lock<std::mutex> ul { cb_mutex };

  TRISYCL_DUMP_T("Before write reservation cb.size() = " << cb.size()
                 << " size() = " << size());
  if (s == 0)
    // Empty reservation requested, so nothing to do
    return false;

  if (blocking)
    /* If in blocking mode, wait for enough room in the pipe, that
       may be changed when a read is done. Do not use a difference
       here because it is only about unsigned values */
    read_done.wait(ul, [&] { return cb.size() + s <= capacity(); });
  else if (cb.size() + s > capacity())
    // Not enough room in the pipe for the reservation
    return false;

  /* If there is enough room in the pipe, just create default values
       in it to do the reservation */
  for (std::size_t i = 0; i != s; ++i)
    cb.push_back();
  /* Compute the location of the first element a posteriori since it
       may not exist a priori if cb was empty before */
  auto first = cb.end() - s;
  /* Add a description of the reservation at the end of the
     reservation queue */
  w_rid_q.emplace_back(first, s);
  // Return the iterator to the last reservation descriptor
  rid = w_rid_q.end() - 1;
  TRISYCL_DUMP_T("After reservation cb.size() = " << cb.size()
                 << " size() = " << size());
  return true;
}


/** Process the read reservations that are ready to be released in the
    reservation queue
*/
void move_read_reservation_forward() {
  // Lock the pipe to avoid nuisance
  std::lock_guard<std::mutex> lg { cb_mutex };

  for (;;) {
    if (r_rid_q.empty())
      // No pending reservation, so nothing to do
      break;
```

```
      if (!r_rid_q.front().ready)
        /* If the first reservation is not ready to be released, stop
           because it is blocking all the following in the queue
           anyway */
        break;
      // Remove the reservation to be released from the queue
      r_rid_q.pop_front();
      std::size_t n_to_pop;
      if (r_rid_q.empty())
        // If it was the last one, remove all the reservation
        n_to_pop = read_reserved_frozen;
      else
        // Else remove everything up to the next reservation
        n_to_pop =  r_rid_q.front().start - cb.begin();
      // No longer take into account these reserved slots
      read_reserved_frozen -= n_to_pop;
      // Release the elements from the FIFO
      while (n_to_pop--)
        cb.pop_front();
      // Notify the clients waiting for some room to write in the pipe
      read_done.notify_all();
      /* ...and process the next reservation to see if it is ready to
         be released too */
    }
  }


  /** Process the write reservations that are ready to be released in the
      reservation queue
  */
  void move_write_reservation_forward() {
    // Lock the pipe to avoid nuisance
    std::lock_guard<std::mutex> lg { cb_mutex };

    for (;;) {
      if (w_rid_q.empty())
        // No pending reservation, so nothing to do
        break;
      // Get the first reservation
      const auto &rid = w_rid_q.front();
      if (!rid.ready)
        /* If the reservation is not ready to be released, stop
           because it is blocking all the following in the queue
           anyway */
        break;
      // Remove the reservation to be released from the queue
      w_rid_q.pop_front();
      // Notify the clients waiting to read something from the pipe
      write_done.notify_all();
      /* ...and process the next reservation to see if it is ready to
         be released too */
    }
  }

};

/// @} End the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_HPP
```

11.131 include/CL/sycl/pipe.hpp File Reference

```
#include <cstddef>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
```

`#include "CL/sycl/pipe/detail/pipe.hpp"`
Include dependency graph for pipe.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::pipe< T >

  A SYCL pipe. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.132 pipe.hpp

```
00001 #ifndef TRISYCL_SYCL_PIPE_HPP
00002 #define TRISYCL_SYCL_PIPE_HPP
00003
00004 /** \file The OpenCL SYCL pipe<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013 #include <memory>
```

```

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A SYCL pipe

    Implement a FIFO-style object that can be used through accessors
    to send some objects T from the input to the output
*/
template <typename T>
class pipe
    /* Use the underlying pipe implementation that can be shared in
       the SYCL model */
  : public detail::shared_ptr_implementation<pipe<T>, detail::pipe<T>>,
    detail::debug<pipe<T>> {

  // The type encapsulating the implementation
  using implementation_t = typename pipe::shared_ptr_implementation;

  // Allows the comparison operation to access the implementation
  friend implementation_t;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /// The STL-like types
  /* Since a pipe element cannot be directly addressed without
     accessor, only define value_type here */
  using value_type = T;


  /// Construct a pipe able to store up to capacity T objects
  pipe(std::size_t capacity)
    : implementation_t { new detail::pipe<T> { capacity } } { }


  /** Get an accessor to the pipe with the required mode

      \param Mode is the requested access mode

      \param Target is the type of pipe access required

      \param[in] command_group_handler is the command group handler in
      which the kernel is to be executed
  */
  template <access::mode Mode,
            access::target Target = access::target::pipe>
  accessor<value_type, 1, Mode, Target>
  get_access(handler &command_group_handler) {
    static_assert(Target == access::target::pipe
                  || Target == access::target::blocking_pipe,
                  "get_access(handler) with pipes can only deal with "
                  "access::pipe or access::blocking_pipe");
    return { implementation, command_group_handler };
  }


  /// Return the maximum number of elements that can fit in the pipe
  std::size_t capacity() const {
    return implementation->capacity();
    }

};

/// @} End the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
```

```
00100 */
00101
00102 #endif // TRISYCL_SYCL_PIPE_HPP
```

11.133 include/CL/sycl/pipe/detail/pipe_accessor.hpp File Reference

```
#include <cstddef>
#include <memory>
#include <type_traits>
#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"
```

Include dependency graph for pipe_accessor.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::detail::pipe_accessor< T, AccessMode, Target >

  The accessor abstracts the way pipe data are accessed inside a kernel. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.134 pipe_accessor.hpp

```
#ifndef TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP
#define TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP

/** \file The OpenCL SYCL pipe accessor<> detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>
#include <type_traits>

#include "CL/sycl/access.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp
"

namespace cl {
namespace sycl {

class handler;

namespace detail {

// Forward declaration of detail::accessor to declare the specialization
template <typename T,
          int Dimensions,
          access::mode Mode,
          access::target Target>
class accessor;
/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** The accessor abstracts the way pipe data are accessed inside a
    kernel
*/
template <typename T,
          access::mode AccessMode,
          access::target Target>
class pipe_accessor :
    public detail::debug<detail::pipe_accessor<T, AccessMode, Target>> {

public:

  static constexpr auto rank = 1;
  static constexpr auto mode = AccessMode;
  static constexpr auto target = Target;

  static constexpr bool blocking =
    (target == cl::sycl::access::target::blocking_pipe);

  /// The STL-like types
  using value_type = T;
  using reference = value_type&;
  using const_reference = const value_type&;

private:

  /// The real pipe implementation behind the hood
  std::shared_ptr<detail::pipe<T>> implementation;

  /** Store the success status of last pipe operation

      It is not impacted by reservation success.

```

```
      It does exist even if the pipe accessor is not evaluated in a
      boolean context for, but a use-def analysis can optimise it out
      in that case and not use some storage

      Use a mutable state here so that it can work with a [=] lambda
      capture without having to declare the whole lambda as mutable
  */
  bool mutable ok = false;

public:

  /** Construct a pipe accessor from an existing pipe
   */
  pipe_accessor(const std::shared_ptr<detail::pipe<T>> &
p,
                handler &command_group_handler) :
    implementation { p } {
    //    TRISYCL_DUMP_T("Create a kernel pipe accessor write = "
    //                   << is_write_access());
    // Verify that the pipe is not already used in the requested mode
    if (mode == access::mode::write)
      if (implementation->used_for_writing)
        /// \todo Use pipe_exception instead
        throw std::logic_error { "The pipe is already used for writing." };
      else
        implementation->used_for_writing = true;
    else
      if (implementation->used_for_reading)
        throw std::logic_error { "The pipe is already used for reading." };
      else
        implementation->used_for_reading = true;
  }


  pipe_accessor() = default;


  /// Return the maximum number of elements that can fit in the pipe
  std::size_t capacity() const {
    return implementation->capacity();
  }

  /** Get the current number of elements in the pipe

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement (for
      example on FPGA).
   */
  std::size_t size() const {
    return implementation->size_with_lock();
  }


  /** Test if the pipe is empty

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      write side (for example on FPGA).
   */
  bool empty() const {
    return implementation->empty_with_lock();
  }


  /** Test if the pipe is full

      This is obviously a volatile value which is constrained by
      restricted relativity.

      Note that on some devices it may be costly to implement on the
      read side (for example on FPGA).
  */
  bool full() const {
    return implementation->full_with_lock();
  }


  /** In an explicit bool context, the accessor gives the success
      status of the last access

      It is not impacted by reservation success.

      The explicitness is related to avoid \code some_pipe <<
```

```
      some_value \endcode to be interpreted as \code some_bool <<
      some_value \endcode when the type of \code some_value \endcode
      is not the same type as the pipe type.

      \return true on success of the previous read or write operation
  */
  explicit operator bool() const {
    return ok;
  }


  /** Try to write a value to the pipe

      \param[in] value is what we want to write

      \return this so we can apply a sequence of write for example
      (but do not do this on a non blocking pipe...)

      \todo provide a && version

      This function is const so it can work when the accessor is
      passed by copy in the [=] kernel lambda, which is not mutable by
      default
  */
  const pipe_accessor &write(const value_type &value) const {
    static_assert(mode == access::mode::write,
                  "'.write(const value_type &value)' method on a pipe accessor"
                  " is only possible with write access mode");
    ok = implementation->write(value, blocking);
    // Return a reference to *this so we can apply a sequence of write
    return *this;
  }


  /** Some syntactic sugar to use \code a << v \endcode instead of
      \code a.write(v) \endcode */
  const pipe_accessor &operator<<(const value_type &value) const {
    static_assert(mode == access::mode::write,
                  "'<<' operator on a pipe accessor is only possible"
                  " with write access mode");
    // Return a reference to *this so we can apply a sequence of >>
    return write(value);
  }


  /** Try to read a value from the pipe

      \param[out] value is the reference to where to store what is
      read

      \return \code this \endcode so we can apply a sequence of read
      for example (but do not do this on a non blocking pipe...)

      This function is const so it can work when the accessor is
      passed by copy in the [=] kernel lambda, which is not mutable by
      default
  */
  const pipe_accessor &read(value_type &value) const {
    static_assert(mode == access::mode::read,
                  "'.read(value_type &value)' method on a pipe accessor"
                  " is only possible with read access mode");
    ok = implementation->read(value, blocking);
    // Return a reference to *this so we can apply a sequence of read
    return *this;
  }


  /** Read a value from a blocking pipe

      \return the read value directly, since it cannot fail on
      blocking pipe

      This function is const so it can work when the accessor is
      passed by copy in the [=] kernel lambda, which is not mutable by
      default
  */
  value_type read() const {
    static_assert(mode == access::mode::read,
                  "'.read()' method on a pipe accessor is only possible"
                  " with read access mode");
    static_assert(blocking,
                  "'.read()' method on a pipe accessor is only possible"
                  " with a blocking pipe");
    value_type value;
    implementation->read(value, blocking);
    return value;
  }
```

```


  /** Some syntactic sugar to use \code a >> v \endcode instead of
      \code a.read(v) \endcode */
  const pipe_accessor &operator>>(value_type &value) const {
    static_assert(mode == access::mode::read,
                  "'>>' operator on a pipe accessor is only possible"
                  " with read access mode");
    // Return a reference to *this so we can apply a sequence of >>
    return read(value);
  }


  detail::pipe_reservation<pipe_accessor>
reserve(std::size_t size) const {
    return { *implementation, size };
  }


  /// Set debug mode
  void set_debug(bool enable) const {
    implementation->debug_mode = enable;
  }


  auto &get_pipe_detail() {
    return implementation;
  }


  ~pipe_accessor() {
    /// Free the pipe for a future usage for the current mode
    if (mode == access::mode::write)
      implementation->used_for_writing = false;
    else
      implementation->used_for_reading = false;
  }

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_DETAIL_PIPE_ACCESSOR_HPP
```

11.135 include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp File Reference

```
#include <cstddef>
#include <memory>
#include "CL/sycl/pipe/detail/pipe.hpp"
```

Include dependency graph for pipe_reservation.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::accessor< T, Dimensions, Mode, Target >

  The buffer accessor abstracts the way buffer data are accessed inside a kernel in a multidimensional variable length array way. More...
- class cl::sycl::detail::pipe_reservation< PipeAccessor >

  The implementation of the pipe reservation station. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.136 pipe_reservation.hpp

```
#ifndef TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP
#define TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP

/** \file The OpenCL SYCL pipe reservation detail behind the scene

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#include "CL/sycl/pipe/detail/pipe.hpp"

namespace cl {
namespace sycl {
namespace detail {

template <typename T,
          int Dimensions,
          access::mode Mode,
          access::target Target>
class accessor;

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/// The implementation of the pipe reservation station
template <typename PipeAccessor>
class pipe_reservation :
    public detail::debug<detail::pipe_reservation<PipeAccessor>> {
  using accessor_type = PipeAccessor;
  static constexpr bool blocking =
    (accessor_type::target ==
cl::sycl::access::target::blocking_pipe);
  using value_type = typename accessor_type::value_type;
  using reference = typename accessor_type::reference;

public:

  using iterator =
    typename detail::pipe<value_type>::implementation_t::iterator
;
  using const_iterator =
    typename detail::pipe<value_type>::implementation_t::const_iterator
;

  // \todo Add to the specification
  static constexpr access::mode mode = accessor_type::mode;
  static constexpr access::target target =
accessor_type::target;

  /**  True if the reservation was successful and still uncommitted. B
       default a pipe_reservation is not reserved and cannot be
       committed */
  bool ok = false;

  /// Point into the reservation buffer. Only valid if ok is true
  typename detail::pipe<value_type>::rid_iterator
rid;

  /** Keep a reference on the pipe to access to the data and methods

      Note that with inlining and CSE it should not use more register
      when compiler optimization is in use. */
  detail::pipe<value_type> &p;


  /** Test that the reservation is in a usable state

      \todo Throw exception instead
  */
  void assume_validity() {
    assert(ok);
  }

public:

  /// Create a pipe reservation station that reserves the pipe itself
  pipe_reservation(detail::pipe<value_type> &p, std::size_t s) : p
{ p } {
```

```
    static_assert(mode == access::mode::write
                  || mode == access::mode::read,
                  "A pipe can only be accesed in read or write mode,"
                  " exclusively");

    /* Since this test is constexpr and dependent of a template
       parameter, it should be equivalent to a specialization of the
       method but in a clearer way */
    if (mode == access::mode::write)
      ok = p.reserve_write(s, rid, blocking);
    else
      ok = p.reserve_read(s, rid, blocking);
  }


  /** No copy constructor with some spurious commit in the destructor
      of the original object
  */
  pipe_reservation(const pipe_reservation &) = delete;


  /// Only a move constructor is required to move it into the shared_ptr
  pipe_reservation(pipe_reservation &&orig) :
    ok {orig.ok },
    rid {orig.rid },
    p { orig.p } {
      /* Even when an object is moved, the destructor of the old
         object is eventually called, so leave the old object in a
         destructable state but without any commit capability */
      orig.ok = false;
    }


  /** Keep the default constructors too

      Otherwise there is no move semantics and the copy is made by
      creating a new reservation and destructing the old one with a
      spurious commit in the meantime...
  */
  pipe_reservation() = default;


  /** Test if the reservation succeeded and thus if the reservation
      can be committed

      Note that it is up to the user to ensure that all the
      reservation elements have been initialized correctly in the case
      of a write for example
  */
  operator bool() {
    return ok;
  }


  /// Start of the reservation area
  iterator begin() {
    assume_validity();
    return rid->start;
  }


  /// Past the end of the reservation area
  iterator end() {
    assume_validity();
    return rid->start + rid->size;
  }


  /// Get the number of elements in the reservation station
  std::size_t size() {
    assume_validity();
    return rid->size;
  }


  /// Access to an element of the reservation
  reference operator[](std::size_t index) {
    assume_validity();
    TRISYCL_DUMP_T("[] index = " << index
                   << " Reservation write address = " << &(rid->start[index]));

    return rid->start[index];
  }


  /** Commit the reservation station

```

```
      \todo Add to the specification that for simplicity a reservation
      can be commited several times but only the first one is taken
      into account
  */
  void commit() {
    if (ok) {
      // If the reservation is in a committable state, commit
      TRISYCL_DUMP_T("Commit");
      rid->ready = true;
      if (mode == access::mode::write)
        p.move_write_reservation_forward();
      else
        p.move_read_reservation_forward();
      ok = false;
    }
  }


  /// An implicit commit is made in the destructor
  ~pipe_reservation() {
    commit();
  }

};

/// @} End the data Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_RESERVATION_DETAIL_PIPE_RESERVATION_HPP
```

11.137 include/CL/sycl/pipe_reservation.hpp File Reference

```
#include <cstddef>
#include <iterator>
#include <memory>
#include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp"
```

Include dependency graph for pipe_reservation.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::pipe_reservation< PipeAccessor >

  The pipe reservation station allows to reserve an array-like view inside the pipe for ordered race-free access from various work-items for example. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.138 pipe_reservation.hpp

```
00001 #ifndef TRISYCL_SYCL_PIPE_RESERVATION_HPP
00002 #define TRISYCL_SYCL_PIPE_RESERVATION_HPP
00003
00004 /** \file The reservation station for OpenCL SYCL pipe accessor<>
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011
00012 #include <cstddef>
00013 #include <iterator>
00014 #include <memory>
00015
00016 #include "CL/sycl/pipe_reservation/detail/pipe_reservation.hpp
      "
00017
00018 namespace cl {
00019 namespace sycl {
00020
00021 /** \addtogroup data Data access and storage in SYCL
00022     @{
00023 */
00024
00025 /** The pipe reservation station allows to reserve an array-like view
00026     inside the pipe for ordered race-free access from various
00027     work-items for example
00028  */
```

```
template <typename PipeAccessor>
struct pipe_reservation {
  using accessor_type = PipeAccessor;
  static constexpr bool blocking =
    (accessor_type::target ==
cl::sycl::access::target::blocking_pipe);
  using accessor_detail = typename accessor_type::accessor_detail;
  /// The STL-like types
  using value_type = typename accessor_type::value_type;
  using reference = value_type&;
  using const_reference = const value_type&;
  using pointer = value_type*;
  using const_pointer = const value_type*;
  using size_type = std::size_t;
  using difference_type = ptrdiff_t;
  using iterator =
    typename detail::pipe_reservation<accessor_detail>::iterator
;
  using const_iterator =
    typename detail::pipe_reservation<accessor_detail>::const_iterator
;
  using reverse_iterator = std::reverse_iterator<iterator>;
  using const_reverse_iterator = std::reverse_iterator<const_iterator>;

  /** Point to the underlying implementation that can be shared in the
      SYCL model with a handler semantics */
  typename std::shared_ptr<detail::pipe_reservation<accessor_detail>>
  implementation;

  /** Use default constructors so that we can create a new buffer copy
      from another one, with either a l-value or a r-value (for
      std::move() for example).

      Since we just copy the shared_ptr<> above, this is where/how the
      sharing magic is happening with reference counting in this case.
  */
  pipe_reservation() = default;


  /// Create a pipe_reservation for an accessor and a number of elements
  pipe_reservation(accessor_type &accessor, std::size_t s)
    : implementation {
    new detail::pipe_reservation<accessor_detail> {
      get_pipe_detail(accessor), s }
  } {}


  /** Create a pipe_reservation from the implementation detail

      This is an internal constructor to allow reserve() on the
      implementation to lift a full-fledged object through
      accessor::reserve().

      \todo Make it private and add required friends
   */
  pipe_reservation(detail::pipe_reservation<accessor_detail>
 &&pr)
    : implementation {
    new detail::pipe_reservation<accessor_detail> { std::move(pr)
} }
  {}


  /** Test if the pipe_reservation has been correctly allocated

      \return true if the pipe_reservation can be used and committed
  */
  operator bool() const {
    return *implementation;
  }


  /// Get the number of reserved element(s)
  std::size_t size() const {
    return implementation->size();
  }


  /// Access to a given element of the reservation
  reference operator[](std::size_t index) const {
    return (*implementation)[index];
  }


  /** Force a commit operation

      Normally the commit is implicitly done in the destructor, but
```

```
      sometime it is useful to do it earlier.
  */
  void commit() const {
    return implementation->commit();
  }


  /// Get an iterator on the first element of the reservation station
  iterator begin() const {
    return implementation->begin();
  }


  /// Get an iterator past the end of the reservation station
  iterator end() const {
    return implementation->end();
  }


  /// Build a constant iterator on the first element of the reservation station
  const_iterator cbegin() const {
    return implementation->begin();
  }


  /// Build a constant iterator past the end of the reservation station
  const_iterator cend() const {
    return implementation->end();
  }


  /// Get a reverse iterator on the last element of the reservation station
  reverse_iterator rbegin() const {
    return std::make_reverse_iterator(end());
  }


  /** Get a reverse iterator on the first element past the end of the
      reservation station */
  reverse_iterator rend() const {
    return std::make_reverse_iterator(begin());
  }


  /** Get a constant reverse iterator on the last element of the
      reservation station */
  const_reverse_iterator crbegin() const {
    return std::make_reverse_iterator(cend());
  }


  /** Get a constant reverse iterator on the first element past the
      end of the reservation station */
  const_reverse_iterator crend() const {
    return std::make_reverse_iterator(cbegin());
  }

};

/// @} End the data Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PIPE_RESERVATION_HPP
```

11.139 include/CL/sycl/platform/detail/host_platform.hpp File Reference

```
#include <memory>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
```

```
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
#include "CL/sycl/platform/detail/platform.hpp"
```

Include dependency graph for host_platform.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::host_platform

  SYCL host platform. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.140 host_platform.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP

/** \file The OpenCL triSYCL host platform implementation

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/
#include <memory>

#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/singleton.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/info/platform.hpp"
#include "CL/sycl/platform/detail/platform.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// SYCL host platform
class host_platform : public detail::platform,
                      public detail::singleton<host_platform> {

// \todo Have this compatible with has_extension
auto static constexpr platform_extensions = "Xilinx_blocking_pipes";

public:

#ifdef TRISYCL_OPENCL
  /** Return the cl_platform_id of the underlying OpenCL platform

      This throws an error since there is no OpenCL platform associated
      to the host platform.
  */
  cl_platform_id get() const override {
    throw non_cl_error("The host platform has no OpenCL platform");
  }


  /** Return the underlying Boost.Compute platform

      This throws an error since there is no Boost Compute platform associated
      to the host platform.
   */
  boost::compute::platform &get_boost_compute() const override {
    throw
      non_cl_error("The host device has no underlying Boost Compute platform");
  }
#endif

```

```

  /// Return true since this platform is the SYCL host platform
  bool is_host() const override {
    return true;
  }


  /** Returning the information parameters for the host platform
      implementation
  */
  string_class get_info_string(info::platform param) const
override {
    switch (param) {
    case info::platform::profile:
      /* Well... Is the host platform really a full profile whereas it
         is not really OpenCL? */
      return "FULL_PROFILE";

    case info::platform::version:
      // \todo I guess it should include the software version too...
      return "2.2";

    case info::platform::name:
      return "triSYCL host platform";

    case info::platform::vendor:
      return "triSYCL Open Source project";

    case info::platform::extensions:
      return platform_extensions;

    default:
      // \todo Define some SYCL exception type for this type of errors
      throw std::invalid_argument {
        "Unknown parameter value for SYCL platform information" };
    }
  }


  /** Specify whether a specific extension is supported on the platform

      \todo To be implemented
  */
  bool has_extension(const string_class &extension) const override {
    detail::unimplemented();
    return {};
  }

  /** Get all the available devices for the host platform

      \param[in] device_type is the device type to filter the selection
      or \c info::device_type::all by default to return all the
      devices

      \return the device list
  */
  vector_class<cl::sycl::device>
  get_devices(info::device_type device_type) const override;

};

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
```

11.141 include/CL/sycl/platform/detail/host_platform_tail.hpp File Reference

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.142 host_platform_tail.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_TAIL_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_TAIL_HPP

/** \file The ending part of the SYCL host platform

    This is here to break a dependency between platform and device

    a-doumoulakis at gmail dot com

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Get all the available devices for this platform

    \param[in] device_type is the device type to filter the selection
    or \c info::device_type::all by default to return all the
    devices

    \return the device list
*/
vector_class<cl::sycl::device>
inline host_platform::get_devices(info::device_type device_type)
 const {
  /** If \c get_devices is called with the host platform
      and the right device type, returns the host_device.
  */
  if (device_type_selector { device_type }(cl::sycl::device {}) > 0)
```

```
    // Return 1 default device, i.e. the host device
    return { {} };
  else
    // No matching device
    return {};
}

/// @} to end the Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_TAIL_HPP
```

11.143 include/CL/sycl/platform/detail/opencl_platform.hpp File Reference

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform/detail/platform.hpp"
```

Include dependency graph for opencl_platform.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::opencl_platform

  SYCL OpenCL platform. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.144 opencl_platform.hpp

```
00001 #ifndef TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_HPP
00002 #define TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_HPP
00003
00004 /** \file The OpenCL triSYCL OpenCL platform implementation
00005
00006     Ronan at Keryell point FR
00007
00008     This file is distributed under the University of Illinois Open Source
00009     License. See LICENSE.TXT for details.
00010 */
00011 #include <memory>
00012
00013 #include <boost/compute.hpp>
```

```
#include "CL/sycl/detail/default_classes.hpp"

#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/platform/detail/platform.hpp"

namespace cl {
namespace sycl {

class device;

namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/// SYCL OpenCL platform
class opencl_platform : public detail::platform {

  /// Use the Boost Compute abstraction of the OpenCL platform
  boost::compute::platform p;

  /** A cache to always return the same live platform for a given OpenCL
      platform

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_platform_id, detail::opencl_platform> cache;

public:

  /// Return the cl_platform_id of the underlying OpenCL platform
  cl_platform_id get() const override {
    return p.id();
  }


  /// Return the underlying Boost.Compute platform
  const boost::compute::platform &get_boost_compute() const override {
    return p;
  }


  /// Return false since an OpenCL platform is not the SYCL host platform
  bool is_host() const override {
    return false;
  }


  /// Returning the information string parameters for the OpenCL platform
  string_class get_info_string(info::platform param) const override {
    /* Use the fact that the triSYCL info values are the same as the
       OpenCL ones used in Boost.Compute to just cast the enum class
       to the int value */
    return p.get_info<std::string>(static_cast<cl_platform_info>(param));
  }


  /// Specify whether a specific extension is supported on the platform
  bool has_extension(const string_class &extension) const override {
    return p.supports_extension(extension);
  }


  ///// Get a singleton instance of the opencl_platform
  static std::shared_ptr<opencl_platform>
  instance(const boost::compute::platform &p) {
    return cache.get_or_register(p.id(),
                                 [&] { return new opencl_platform { p }; });
  }


  /** Get all the available devices for this OpenCL platform

      \param[in] device_type is the device type to filter the selection
      or \c info::device_type::all by default to return all the
      devices

      \return the device list
```

```
  */
  vector_class<cl::sycl::device>
  get_devices(info::device_type device_type) const override;

private:

  /// Only the instance factory can built it
  opencl_platform(const boost::compute::platform &p) : p { p } {}

public:

  /// Unregister from the cache on destruction
  ~opencl_platform() override {
    cache.remove(p.id());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
TRISYCL_WEAK_ATTRIB_PREFIX
detail::cache<cl_platform_id, detail::opencl_platform>
 opencl_platform::cache
TRISYCL_WEAK_ATTRIB_SUFFIX;

/// @} to end the execution Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_HOST_PLATFORM_HPP
```

11.145 include/CL/sycl/platform/detail/opencl_platform_tail.hpp File Reference

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.146 opencl_platform_tail.hpp

```
#ifndef TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_TAIL_HPP
#define TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_TAIL_HPP

/** \file The ending part of the SYCL host platform

    This is here to break a dependency between platform and device

    a-doumoulakis at gmail dot com

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

namespace cl {
namespace sycl {
namespace detail {

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

/** Returns a vector class containing all SYCL devices associated with
    this OpenCL platform

    \param[in] device_type is the device type to filter the selection
    or \c info::device_type::all by default to return all the
    devices

    \return the device list
*/
vector_class<cl::sycl::device>
inline opencl_platform::get_devices(
info::device_type device_type) const {
  vector_class<cl::sycl::device> devices;
    device_type_selector ds { device_type };
  // Add the desired OpenCL devices
  for (const auto &d : get_boost_compute().devices()) {
    // Get the SYCL device from the Boost Compute device
    cl::sycl::device sycl_dev { d };
    /* Return the devices with the good criterion according to the selector.
       By calling devices on the \c boost::compute::platform we know that
       we iterate only over the device belonging to the current platform,
    */
    if (ds(sycl_dev) > 0)
      devices.push_back(sycl_dev);
  }

  return devices;
}

/// @} to end the Doxygen group

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_PLATFORM_DETAIL_OPENCL_PLATFORM_TAIL_HPP
```

11.147 include/CL/sycl/queue/detail/host_queue.hpp File Reference

```
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
```

```
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for host_queue.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::host_queue

  Some implementation details about the SYCL queue.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.148 host_queue.hpp

```
#ifndef TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP

/** \file Some implementation details of the host queue

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** Some implementation details about the SYCL queue

    Note that a host queue is not a singleton, compared to host
    device or host platform, for example.
 */
class host_queue : public detail::queue,
                   detail::debug<host_queue> {

#ifdef TRISYCL_OPENCL
  /** Return the cl_command_queue of the underlying OpenCL queue

      This throws an error since there is no OpenCL queue associated
      to the host queue.
  */
  cl_command_queue get() const override {
    throw non_cl_error("The host queue has no OpenCL command queue");
  }


  /** Return the underlying Boost.Compute command queue

      This throws an error since there is no OpenCL queue associated
      to the host queue.
  */
  boost::compute::command_queue &get_boost_compute() override {
    throw non_cl_error("The host queue has no OpenCL command queue");
  }
#endif


  /// Return the SYCL host queue's host context
  cl::sycl::context get_context() const override {
    // Return the default context which is the host context
    return {};
  }


  /// Return the SYCL host device the host queue is associated with
  cl::sycl::device get_device() const override {
    // Return the default device which is the host device
    return {};
  }


  /// Claim proudly that the queue is executing on the SYCL host device
  bool is_host() const override {
    return true;
  }


};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
```

```
00085     ### eval: (flyspell-prog-mode)
00086     ### End:
00087 */
00088
00089 #endif // TRISYCL_SYCL_QUEUE_DETAIL_HOST_QUEUE_HPP
```

11.149 include/CL/sycl/queue/detail/opencl_queue.hpp File Reference

```
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"
```

Include dependency graph for opencl_queue.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::detail::opencl_queue

  Some implementation details about the SYCL queue.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.150 opencl_queue.hpp

```
#ifndef TRISYCL_SYCL_QUEUE_DETAIL_OPENCL_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_DETAIL_OPENCL_QUEUE_HPP

/** \file Some implementation details of the OpenCL queue

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/cache.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/queue/detail/queue.hpp"

namespace cl {
namespace sycl {
namespace detail {

/// Some implementation details about the SYCL queue
class opencl_queue : public detail::queue,
                     detail::debug<opencl_queue> {
  /// Use the Boost Compute abstraction of the OpenCL command queue
  boost::compute::command_queue q;

  /** A cache to always return the same alive queue for a given OpenCL
      command queue

      C++11 guaranties the static construction is thread-safe
  */
  static detail::cache<cl_command_queue, detail::opencl_queue>
 cache;

  /// Return the cl_command_queue of the underlying OpenCL queue
  cl_command_queue get() const override {
    return q.get();
  }


  /// Return the underlying Boost.Compute command queue
  boost::compute::command_queue &get_boost_compute() override {
    return q;
  }


  /// Return the SYCL context associated to the queue
  cl::sycl::context get_context() const override {
    return q.get_context();
  }


  /// Return the SYCL device associated to the queue
  cl::sycl::device get_device() const override {
    return q.get_device();
  }


  /// Claim proudly that an OpenCL queue cannot be the SYCL host queue
  bool is_host() const override {
    return false;
  }

private:

  /// Only the instance factory can built it
  opencl_queue(const boost::compute::command_queue &q) : q { q } {}

public:
```

```

  /// Get a singleton instance of the opencl_queue
  static std::shared_ptr<opencl_queue>
  instance(const boost::compute::command_queue &q) {
    return cache.get_or_register(q.get(),
                                 [&] { return new opencl_queue { q }; });
  }


  /** Create a new queue associated to this device

      \todo Check with SYCL committee what is the expected behaviour
      here about the context. Is this a new context everytime, or
      always the same for a given device?
  */
  static std::shared_ptr<detail::queue>
  instance(const cl::sycl::device &d) {
    return instance (boost::compute::command_queue {
        // For now, create a new context every time
        boost::compute::context { d.get_boost_compute() },
        d.get_boost_compute()
          });
  }


  /// Unregister from the cache on destruction
  ~opencl_queue() override {
    cache.remove(q.get());
  }

};

/* Allocate the cache here but since this is a pure-header library,
   use a weak symbol so that only one remains when SYCL headers are
   used in different compilation units of a program
*/
TRISYCL_WEAK_ATTRIB_PREFIX
detail::cache<cl_command_queue, detail::opencl_queue>
opencl_queue::cache
TRISYCL_WEAK_ATTRIB_SUFFIX;

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_QUEUE_DETAIL_OPENCL_QUEUE_HPP
```

11.151 include/CL/sycl/queue/detail/queue.hpp File Reference

```
#include <atomic>
#include <condition_variable>
#include <mutex>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/detail/debug.hpp"
```

Include dependency graph for queue.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::detail::queue

  Some implementation details about the SYCL queue.

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::detail

11.152 queue.hpp

```
#ifndef TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP

/** \file Some implementation details of queue.

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <atomic>
#include <condition_variable>
#include <mutex>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/context.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/detail/debug.hpp"

namespace cl {
namespace sycl {
namespace detail {

/** Some implementation details about the SYCL queue
 */
struct queue : detail::debug<detail::queue> {
  /// Track the number of kernels still running to wait for their completion
  std::atomic<size_t> running_kernels;

  /// To signal when all the kernels have completed
  std::condition_variable finished;
  /// To protect the access to the condition variable
  std::mutex finished_mutex;


  /// Initialize the queue with 0 running kernel
  queue() {
    running_kernels = 0;
  }


  /// Wait for all kernel completion
  void wait_for_kernel_execution() {
    TRISYCL_DUMP_T("Queue waiting for kernel completion");
    std::unique_lock<std::mutex> ul { finished_mutex };
    finished.wait(ul, [&] {
        // When there is no kernel running in this queue, we are ready to go
        return running_kernels == 0;
      });
  }


  /// Signal that a new kernel started on this queue
  void kernel_start() {
    TRISYCL_DUMP_T("A kernel has been added to the queue");
    // One more kernel
    ++running_kernels;
  }


  /// Signal that a new kernel finished on this queue
  void kernel_end() {
    TRISYCL_DUMP_T("A kernel of the queue ended");
    if (--running_kernels == 0) {
      /* It was the last kernel running, so signal the queue just in
         case it was working for it for completion

         In some cases several threads might want to wait for the
         same queue, because of this \c notify_one is not be enough
         and a \c notify_all is needed
      */
      finished.notify_all();
    }
  }


#ifdef TRISYCL_OPENCL
  /** Return the underlying OpenCL command queue after doing a retain

      This memory object is expected to be released by the developer.
```

```

    Retain a reference to the returned cl_command_queue object.

    Caller should release it when finished.

    If the queue is a SYCL host queue then an exception is thrown.
  */
  virtual cl_command_queue get() const = 0;

  /// Return the underlying Boost.Compute command queue
  virtual boost::compute::command_queue &get_boost_compute() = 0;
#endif


  /** Return the SYCL queue's context

    Report errors using SYCL exception classes.
  */
  virtual cl::sycl::context get_context() const = 0;


  /** Return the SYCL device the queue is associated with

    Report errors using SYCL exception classes.
  */
  virtual cl::sycl::device get_device() const = 0;


  /// Return whether the queue is executing on a SYCL host device
  virtual bool is_host() const = 0;


  /// Wait for all kernel completion before the queue destruction
  /// \todo Update according spec since queue destruction is non blocking
  virtual ~queue() {
    wait_for_kernel_execution();
  }

};

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_QUEUE_DETAIL_QUEUE_HPP
```

11.153 include/CL/sycl/queue.hpp File Reference

```
#include <memory>
#include <boost/compute.hpp>
#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/handler_event.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/queue/detail/host_queue.hpp"
#include "CL/sycl/queue/detail/opencl_queue.hpp"
```

Include dependency graph for queue.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::queue

  SYCL queue, similar to the OpenCL queue concept. More...
- struct std::hash< cl::sycl::queue >

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::info
- std

Typedefs

- using cl::sycl::info::queue_profiling = bool

Enumerations

- enum cl::sycl::info::queue : int { cl::sycl::info::queue::context, cl::sycl::info::queue::device, cl::sycl::info↩ ::queue::reference_count, cl::sycl::info::queue::properties }

  Queue information descriptors.

11.154 queue.hpp

```
#ifndef TRISYCL_SYCL_QUEUE_HPP
#define TRISYCL_SYCL_QUEUE_HPP

/** \file The OpenCL SYCL queue

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <memory>

#ifdef TRISYCL_OPENCL
#include <boost/compute.hpp>
#endif

#include "CL/sycl/context.hpp"
#include "CL/sycl/detail/debug.hpp"
#include "CL/sycl/detail/default_classes.hpp"
#include "CL/sycl/detail/unimplemented.hpp"
#include "CL/sycl/device.hpp"
#include "CL/sycl/device_selector.hpp"
#include "CL/sycl/exception.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/handler_event.hpp"
#include "CL/sycl/info/param_traits.hpp"
#include "CL/sycl/parallelism.hpp"
#include "CL/sycl/queue/detail/host_queue.hpp"
#ifdef TRISYCL_OPENCL
#include "CL/sycl/queue/detail/opencl_queue.hpp"
#endif

namespace cl {
namespace sycl {

class context;
class device_selector;

/** \addtogroup execution Platforms, contexts, devices and queues
    @{
*/

namespace info {

using queue_profiling = bool;

/** Queue information descriptors

    From specification C.4

    \todo unsigned int?

    \todo To be implemented
*/
enum class queue : int {
  context,
  device,
  reference_count,
  properties
};

/** Dummy example for get_info() on queue::context that would return a
    context

    \todo Describe all the types
*/
TRISYCL_INFO_PARAM_TRAITS(queue::context,
context)

}
```

```


/** SYCL queue, similar to the OpenCL queue concept.

    \todo The implementation is quite minimal for now. :-)

    \todo All the queue methods should return a queue& instead of void
    to it is possible to chain opoerations
*/
class queue
    /* Use the underlying queue implementation that can be shared in
       the SYCL model */
  : public detail::shared_ptr_implementation<queue, detail::queue>,
    detail::debug<queue> {

  // The type encapsulating the implementation
  using implementation_t = typename queue::shared_ptr_implementation;

  /* Allows the comparison operation to sneak in

     Required from Clang++ 3.9 and G++ 6
  */
  friend implementation_t;

public:

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  /** Default constructor for platform which is the host platform

      Returns errors via the SYCL exception class.

      \todo Check with the specification if it is the host queue or
      the one related to the default device selector.
  */
  queue() : implementation_t { new detail::host_queue } {}


  /** This constructor creates a SYCL queue from an OpenCL queue

      At construction it does a retain on the queue memory object.

      Retain a reference to the cl_command_queue object. Caller should
      release the passed cl_command_queue object when it is no longer
      needed.

      Return synchronous errors regarding the creation of the queue and
      report asynchronous errors via the async_handler callback function
      in conjunction with the synchronization and throw methods.

      Note that the default case asyncHandler = nullptr is handled by the
      default constructor.

  */
  explicit queue(async_handler asyncHandler) : queue { } {
    detail::unimplemented();
  }


  /** Creates a queue for the device provided by the device selector

      If no device is selected, an error is reported.

      Return synchronous errors regarding the creation of the queue and
      report asynchronous errors via the \c async_handler callback
      function if and only if there is an \c async_handler provided.
  */
  queue(const device_selector &deviceSelector,
        async_handler asyncHandler = nullptr)
    // Just create the queue from the selected device
    : queue { device { deviceSelector }, asyncHandler} { }


  /** A queue is created for a SYCL device

      Return asynchronous errors via the \c async_handler callback
      function.
  */
  queue(const device &d,
        async_handler asyncHandler = nullptr) : implementation_t {
#ifdef TRISYCL_OPENCL
    d.is_host()
      ? std::shared_ptr<detail::queue>{ new detail::host_queue }
      : detail::opencl_queue::instance(d)
#else
```

```
    new detail::host_queue
#endif

  } { }


  /** This constructor chooses a device based on the provided
      device_selector, which needs to be in the given context.

      If no device is selected, an error is reported.

      Return synchronous errors regarding the creation of the queue.

      If and only if there is an asyncHandler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const context &syclContext,
        const device_selector &deviceSelector,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


  /** Creates a command queue using clCreateCommandQueue from a context
      and a device

      Return synchronous errors regarding the creation of the queue.

      If and only if there is an asyncHandler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const context &syclContext,
        const device &syclDevice,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


  /** Creates a command queue using clCreateCommandQueue from a context
      and a device

      It enables profiling on the queue if the profilingFlag is set to
      true.

      Return synchronous errors regarding the creation of the queue. If
      and only if there is an asyncHandler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const context &syclContext,
        const device &syclDevice,
        info::queue_profiling profilingFlag,
        async_handler asyncHandler = nullptr) : queue { } {
    detail::unimplemented();
  }


#ifdef TRISYCL_OPENCL
  /** This constructor creates a SYCL queue from an OpenCL queue

      At construction it does a retain on the queue memory object.

      Return synchronous errors regarding the creation of the queue. If
      and only if there is an async_handler provided, it reports
      asynchronous errors via the async_handler callback function in
      conjunction with the synchronization and throw methods.
  */
  queue(const cl_command_queue &q, async_handler ah = nullptr)
    : queue { boost::compute::command_queue { q }, ah } {}


  /** Construct a queue instance using a boost::compute::command_queue

      This is a triSYCL extension for boost::compute interoperation.

      Return synchronous errors via the SYCL exception class.

      \todo Deal with handler
  */
  queue(const boost::compute::command_queue &q, async_handler ah = nullptr)
    : implementation_t { detail::opencl_queue::instance(q) }
{}
#endif
```

```
#ifdef TRISYCL_OPENCL
  /** Return the underlying OpenCL command queue after doing a retain

      This memory object is expected to be released by the developer.

      Retain a reference to the returned cl_command_queue object.

      Caller should release it when finished.

      If the queue is a SYCL host queue then an exception is thrown.
  */
  cl_command_queue get() const {
    return implementation->get();
  }


  /** Return the underlying Boost.Compute command queue if it is an
      OpenCL queue

      This is a triSYCL extension
  */
  boost::compute::command_queue get_boost_compute() const {
    return implementation->get_boost_compute();
  }
#endif


  /** Return the SYCL queue's context

      Report errors using SYCL exception classes.
  */
  context get_context() const {
    return implementation->get_context();
  }


  /** Return the SYCL device the queue is associated with

      Report errors using SYCL exception classes.
  */
  device get_device() const {
    return implementation->get_device();
  }


  /// Return whether the queue is executing on a SYCL host device
  bool is_host() const {
    return implementation->is_host();
  }


  /** Performs a blocking wait for the completion all enqueued tasks in
      the queue

      Synchronous errors will be reported through SYCL exceptions.
  */
  void wait() {
    implementation->wait_for_kernel_execution();
  }


  /** Perform a blocking wait for the completion all enqueued tasks in the queue

      Synchronous errors will be reported via SYCL exceptions.

      Asynchronous errors will be passed to the async_handler passed to the
      queue on construction.

      If no async_handler was provided then asynchronous exceptions will
      be lost.
  */
  void wait_and_throw() {
    // \todo Implement the throw part of wait_and_throw
    wait();
    detail::unimplemented();
  }


  /** Checks to see if any asynchronous errors have been produced by the
      queue and if so reports them by passing them to the async_handler
      passed to the queue on construction

      If no async_handler was provided then asynchronous exceptions will
      be lost.
  */
  void throw_asynchronous() {
    detail::unimplemented();
```

```
  }


  /// Queries the platform for cl_command_queue info
  template <info::queue param>
  typename info::param_traits<info::queue, param>::type get_info() const  {
    detail::unimplemented();
    return {};
  }


  /** Submit a command group functor to the queue, in order to be
      scheduled for execution on the device

      Use an explicit functor parameter taking a handler& so we can use
      "auto" in submit() lambda parameter.

      \todo Add in the spec an implicit conversion of handler_event to
      queue& so it is possible to chain operations on the queue
  */
  template <typename Handler_Functor>
  handler_event submit(Handler_Functor cgf) {
    handler command_group_handler { implementation };
    cgf(command_group_handler);
    return {};
  }


  /** Submit a command group functor to the queue, in order to be
      scheduled for execution on the device

      On kernel error, this command group functor, then it is scheduled
      for execution on the secondary queue.

      Return a command group functor event, which is corresponds to the
      queue the command group functor is being enqueued on.
  */
  handler_event submit(std::function<void(handler &)> cgf, queue &secondaryQueue) {
    detail::unimplemented();
    // Since it is not implemented, always submit on the main queue
    return submit(cgf);
  }

};

/// @} to end the execution Doxygen group

}
}

/* Inject a custom specialization of std::hash to have the buffer
   usable into an unordered associative container

   \todo Add this to the spec
*/
namespace std {

template <> struct hash<cl::sycl::queue> {

  auto operator()(const cl::sycl::queue &q) const {
    // Forward the hashing to the implementation
    return q.hash();
  }

};

}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_QUEUE_HPP
```

11.155 include/CL/sycl/range.hpp File Reference

```
#include <functional>
```

```
#include <numeric>
#include "CL/sycl/detail/small_array.hpp"
```

Include dependency graph for range.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::range< Dimensions >

  A SYCL range defines a multi-dimensional index range that can be used to define launch parallel computation extent or buffer sizes. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Functions

- auto cl::sycl::make_range (range< 1 > r)

  Implement a make_range to construct a range<> of the right dimension with implicit conversion from an initializer list for example.
- auto cl::sycl::make_range (range< 2 > r)
- auto cl::sycl::make_range (range< 3 > r)
- template<typename... BasicType>
  auto cl::sycl::make_range (BasicType... Args)

  Construct a range<> from a function call with arguments, like make_range(1, 2, 3)

11.156 range.hpp

```
#ifndef TRISYCL_SYCL_RANGE_HPP
#define TRISYCL_SYCL_RANGE_HPP

/** \file The OpenCL SYCL range<>

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <functional>
#include <numeric>
#include "CL/sycl/detail/small_array.hpp"

namespace cl {
namespace sycl {

/** \addtogroup parallelism Expressing parallelism through kernels
    @{
*/

/** A SYCL range defines a multi-dimensional index range that can be used
    to define launch parallel computation extent or buffer sizes.

    \todo use std::size_t Dimensions instead of int Dimensions in the
        specification?

    \todo add to the specification this default parameter value?

    \todo add to the specification some way to specify an offset?
*/
template <int Dimensions = 1>
class range : public detail::small_array_123<
                std::size_t,
                range<Dimensions>,
                Dimensions > {

public:

  // Inherit of all the constructors
  using detail::small_array_123<std::size_t,
                                range<Dimensions>,
                                Dimensions>::small_array_123;


  /** Return the number of elements in the range

      \todo Give back size() its real meaning in the specification

      \todo add this method to the specification
  */
  size_t get_count() const {
    // Return the product of the sizes in each dimension
    return std::accumulate(this->cbegin(),
                           this->cend(),
                           1,
                           std::multiplies<size_t> {});
  }
};


/** Implement a make_range to construct a range<> of the right dimension
    with implicit conversion from an initializer list for example.

    Cannot use a template on the number of dimensions because the implicit
    conversion would not be tried.
*/
inline auto make_range(range<1> r) { return r; }
inline auto make_range(range<2> r) { return r; }
inline auto make_range(range<3> r) { return r; }


/** Construct a range<> from a function call with arguments, like
    make_range(1, 2, 3)
*/
template<typename... BasicType>
auto make_range(BasicType... Args) {
  // Call constructor directly to allow narrowing
  return range<sizeof...(Args)>(Args...);
}

/// @} End the parallelism Doxygen group

```

```
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_RANGE_HPP
```

11.157 include/CL/sycl/static_pipe.hpp File Reference

```
#include <cstddef>
#include <memory>
#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"
```

Include dependency graph for static_pipe.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::static_pipe< T, Capacity >

  A SYCL static-scoped pipe equivalent to an OpenCL program-scoped pipe. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

11.158 static_pipe.hpp

```
#ifndef TRISYCL_SYCL_STATIC_PIPE_HPP
#define TRISYCL_SYCL_STATIC_PIPE_HPP

/** \file The OpenCL SYCL static-scoped pipe equivalent to an OpenCL
    program-scoped pipe

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <cstddef>
#include <memory>

#include "CL/sycl/access.hpp"
#include "CL/sycl/accessor.hpp"
#include "CL/sycl/handler.hpp"
#include "CL/sycl/pipe/detail/pipe.hpp"

namespace cl {
namespace sycl {

/** \addtogroup data Data access and storage in SYCL
    @{
*/

/** A SYCL static-scoped pipe equivalent to an OpenCL program-scoped
    pipe

    Implement a FIFO-style object that can be used through accessors
    to send some objects T from the input to the output.

    Compared to a normal pipe, a static_pipe takes a constexpr size
    and is expected to be declared in a compile-unit static context so
    the compiler can generate everything at compile time.

    This is useful to generate a fixed and optimized hardware
    implementation on FPGA for example, where the interconnection
    graph can be also inferred at compile time.

    It is not directly mapped to the OpenCL program-scoped pipe
    because in SYCL there is not this concept of separated
    program. But the SYCL device compiler is expected to generate some
    OpenCL program(s) with program-scoped pipes when a SYCL
    static-scoped pipe is used. These details are implementation
    defined.
*/
template <typename T, std::size_t Capacity>
class static_pipe
    /* Use the underlying pipe implementation that can be shared in
       the SYCL model */
  : public detail::shared_ptr_implementation<static_pipe<T, Capacity>,
                                             detail::pipe<T>>,
    detail::debug<static_pipe<T, Capacity>> {

  // The type encapsulating the implementation
  using implementation_t = typename
static_pipe::shared_ptr_implementation;

  // Make the implementation member directly accessible in this class
  using implementation_t::implementation;

  // Allows the comparison operation to access the implementation
  friend implementation_t;

public:

  /// The STL-like types
  using value_type = T;

```

```

  /// Construct a static-scoped pipe able to store up to Capacity T objects
  static_pipe()
    : implementation_t { new detail::pipe<T> { Capacity } } { }


  /** Get an accessor to the pipe with the required mode

      \param Mode is the requested access mode

      \param Target is the type of pipe access required

      \param[in] command_group_handler is the command group handler in
      which the kernel is to be executed
  */
  template <access::mode Mode,
            access::target Target = access::target::pipe>
  accessor<value_type, 1, Mode, Target>
  get_access(handler &command_group_handler) {
    static_assert(Target == access::target::pipe
                  || Target == access::target::blocking_pipe,
                  "get_access(handler) with pipes can only deal with "
                  "access::pipe or access::blocking_pipe");
    return { implementation, command_group_handler };
  }


  /** Return the maximum number of elements that can fit in the pipe

      This is a constexpr since the capacity is in the type.
  */
  std::size_t constexpr capacity() const {
    return Capacity;
    }

};

/// @} End the execution Doxygen group

}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_STATIC_PIPE_HPP
```

11.159 include/CL/sycl/vec.hpp File Reference

Implement the small OpenCL vector class.

```
#include "CL/sycl/detail/array_tuple_helpers.hpp"
```

Include dependency graph for vec.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- class cl::sycl::vec< DataType, NumElements >

  Small OpenCL vector class. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl

Macros

- #define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) using type##size = vec<actual_type, size>;

  A macro to define type alias, such as for type=uchar, size=4 and actual_type=unsigned char, uchar4 is equivalent to vec<unsigned char, 4>
- #define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)

  Declare the vector types of a type for all the sizes.

11.159.1 Detailed Description

Implement the small OpenCL vector class.

Ronan at Keryell point FR

This file is distributed under the University of Illinois Open Source License. See LICENSE.TXT for details.

Definition in file vec.hpp.

11.160 vec.hpp

```
#ifndef TRISYCL_SYCL_VEC_HPP
#define TRISYCL_SYCL_VEC_HPP

/** \file

    Implement the small OpenCL vector class

    Ronan at Keryell point FR

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include "CL/sycl/detail/array_tuple_helpers.hpp"

namespace cl {
namespace sycl {

/** \addtogroup vector Vector types in SYCL

    @{
*/


/** Small OpenCL vector class

    \todo add [] operator

    \todo add iterators on elements, with begin() and end()

    \todo having vec<> sub-classing array<> instead would solve the
    previous issues

    \todo move the implementation elsewhere

    \todo simplify the helpers by removing some template types since there
    are now inside the vec<> class.

    \todo rename in the specification element_type to value_type
*/
template <typename DataType, size_t NumElements>
class vec : public detail::small_array<DataType,
                                       vec<DataType, NumElements>,
                                       NumElements> {
  using basic_type = typename detail::small_array<DataType,
                                                  vec<DataType, NumElements>,
                                                  NumElements>;

public:
```

```

  /** Construct a vec from anything from a scalar (to initialize all the
      elements with this value) up to an aggregate of scalar and vector
      types (in this case the total number of elements must match the size
      of the vector)
  */
  template <typename... Types>
  vec(const Types... args)
    : basic_type { detail::expand<vec>(flatten_to_tuple<vec>(args...)) } { }


/// Use classical constructors too
  vec() = default;


  // Inherit of all the constructors
  using basic_type::basic_type;

private:

  /** Flattening helper that does not change scalar values but flatten a
      vec<T, n> v into a tuple<T, T,..., T>{ v[0], v[1],..., v[n-1] }

      If we have a vector, just forward its array content since an array
      has also a tuple interface :-) (23.3.2.9 Tuple interface to class
      template array [array.tuple])
  */
  template <typename V, typename Element, size_t s>
  static auto flatten(const vec<Element, s> i) {
    static_assert(s <= V::dimension,
                  "The element i will not fit in the vector");
    return static_cast<std::array<Element, s>>(i);
  }


  /** If we do not have a vector, just forward it as a tuple up to the
      final initialization.

      \return typically tuple<double>{ 2.4 } from 2.4 input for example
  */
  template <typename V, typename Type>
  static auto flatten(const Type i) {
    return std::make_tuple(i);
  }


 /** Take some initializer values and apply flattening on each value

      \return a tuple of scalar initializer values
   */
  template <typename V, typename... Types>
  static auto flatten_to_tuple(const Types... i) {
    // Concatenate the tuples returned by each flattening
    return std::tuple_cat(flatten<V>(i)...);
  }


  /// \todo To implement
#if 0
  vec<dataT,
      numElements>
  operator+(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator-(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator*(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator/(const vec<dataT, numElements> &rhs) const;
  vec<dataT, numElements>
  operator+=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator-=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator*=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator/=(const vec<dataT, numElements> &rhs);
  vec<dataT, numElements>
  operator+(const dataT &rhs) const;
  vec<dataT, numElements>
  operator-(const dataT &rhs) const;
  vec<dataT, numElements>
  operator*(const dataT &rhs) const;
  vec<dataT, numElements>
  operator/(const dataT &rhs) const;
  vec<dataT, numElements>
  operator+=(const dataT &rhs);
  vec<dataT, numElements>
```

```
  operator-=(const dataT &rhs);
  vec<dataT, numElements>
  operator*=(const dataT &rhs);
  vec<dataT, numElements>
  operator/=(const dataT &rhs);
  vec<dataT, numElements> &operator=(const
vec<dataT, numElements> &rhs);
  vec<dataT, numElements> &operator=(const dataT &rhs);
  bool operator==(const vec<dataT, numElements> &rhs) const;
  bool operator!=(const vec<dataT, numElements> &rhs) const;
  // Swizzle methods (see notes)
  swizzled_vec<T, out_dims> swizzle<int s1, ...>();
#ifdef SYCL_SIMPLE_SWIZZLES
  swizzled_vec<T, 4> xyzw();
  ...
#endif // #ifdef SYCL_SIMPLE_SWIZZLES
#endif
};


  /** A macro to define type alias, such as for type=uchar, size=4 and
      actual_type=unsigned char, uchar4 is equivalent to vec<unsigned char, 4>
  */
#define TRISYCL_DEFINE_VEC_TYPE_SIZE(type, size, actual_type) \
  using type##size = vec<actual_type, size>;

  /// Declare the vector types of a type for all the sizes
#define TRISYCL_DEFINE_VEC_TYPE(type, actual_type)           \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 1, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 2, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 3, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 4, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 8, actual_type)         \
  TRISYCL_DEFINE_VEC_TYPE_SIZE(type, 16, actual_type)

  /// Declare all the possible vector type aliases
  TRISYCL_DEFINE_VEC_TYPE(char, char)
  TRISYCL_DEFINE_VEC_TYPE(uchar, unsigned char)
  TRISYCL_DEFINE_VEC_TYPE(short, short int)
  TRISYCL_DEFINE_VEC_TYPE(ushort, unsigned short int)
  TRISYCL_DEFINE_VEC_TYPE(int, int)
  TRISYCL_DEFINE_VEC_TYPE(uint, unsigned int)
  TRISYCL_DEFINE_VEC_TYPE(long, long int)
  TRISYCL_DEFINE_VEC_TYPE(ulong, unsigned long int)
  TRISYCL_DEFINE_VEC_TYPE(float, float)
  TRISYCL_DEFINE_VEC_TYPE(double, double)

/// @} End the vector Doxygen group


}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_VEC_HPP
```

11.161 include/CL/sycl/vendor/Xilinx/opt_decorate_func.hpp File Reference

This graph shows which files directly or indirectly include this file:

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::xilinx

Functions

- template<typename Functor >
  void cl::sycl::xilinx::dataflow (Functor f) noexcept

  Apply dataflow execution on functions or loops.
- template<typename Functor >
  void cl::sycl::xilinx::pipeline (Functor f) noexcept

  Execute loops in a pipelined manner.

11.162 opt_decorate_func.hpp

```
#ifndef TRISYCL_SYCL_VENDOR_XILINX_OPT_DECORATE_FUNC_HPP
#define TRISYCL_SYCL_VENDOR_XILINX_OPT_DECORATE_FUNC_HPP

/** \file This file includes some decorating functions for
    some optimizations supported by Xilinx tools.

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \addtogroup Xilinx Xilinx vendor extensions
    @{
*/

namespace cl {
namespace sycl {
namespace xilinx {

/** Apply dataflow execution on functions or loops

```

```
    With this mode, Xilinx tools analyze the dataflow dependencies
    between sequential functions or loops and create channels (based
    on ping-pong RAMs or FIFOs) that allow consumer functions or loops
    to start operation before the producer functions or loops have
    completed.

    This allows functions or loops to operate in parallel, which
    decreases latency and improves the throughput.

    \param[in] f is a function that functions or loops in f will be executed
    in a dataflow manner.
*/
template <typename Functor>
void dataflow(Functor f) noexcept {
  /* SSDM instruction is inserted before the argument functor to guide xocc to
     do dataflow. */
  _ssdm_op_SpecDataflowPipeline(-1, "");
  f();
}


/** Execute loops in a pipelined manner

    A loop with pipeline mode processes a new input every clock
    cycle. This allows the operations of different iterations of the
    loop to be executed in a concurrent manner to reduce latency.

    \param[in] f is a function with an innermost loop to be executed in a
    pipeline way.
*/
template <typename Functor>
void pipeline(Functor f) noexcept {
  /* SSDM instruction is inserted before the argument functor to guide xocc to
     do pipeline. */
  _ssdm_op_SpecPipeline(1, 1, 0, 0, "");
  f();
}


}
}
}

/// @} End the Xilinx Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_VENDOR_XILINX_OPT_DECORATE_FUNC_HPP
```

11.163 include/CL/sycl/vendor/Xilinx/partition_array.hpp File Reference

```
#include <array>
#include <cstddef>
#include <type_traits>
```

Include dependency graph for partition_array.hpp:

This graph shows which files directly or indirectly include this file:

Classes

- struct cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >

  Represent a cyclic partition.
- struct cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >

  Represent a block partition.
- struct cl::sycl::xilinx::partition::complete< PDim >

  Represent a complete partition.
- struct cl::sycl::xilinx::partition::none

  Represent a none partition.
- struct cl::sycl::xilinx::partition_array< ValueType, Size, PartitionType >

  Define an array class with partition feature. More...

Namespaces

- cl

  The vector type to be used as SYCL vector.
- cl::sycl
- cl::sycl::xilinx
- cl::sycl::xilinx::partition

  Kind of array partition.

Enumerations

- enum cl::sycl::xilinx::partition::type { cl::sycl::xilinx::partition::type::cyclic, cl::sycl::xilinx::partition::type::block, cl::sycl::xilinx::partition::type::complete, cl::sycl::xilinx::partition::type::none }

  Three real partition types: cyclic, block, complete.

11.164 partition_array.hpp

```
#ifndef TRISYCL_SYCL_VENDOR_XILINX_PARTITION_ARRAY_HPP
#define TRISYCL_SYCL_VENDOR_XILINX_PARTITION_ARRAY_HPP

/** \file This is a class expressing arrays that can be partitioned.

    \todo While Xilinx xocc supports multidimensional C arrays, the
    current implementation only support 1-dim array partition.

    \todo Extend this with multidimensional C++ arrays, such as with future
    mdspan C++20 syntax.

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

#include <array>
#include <cstddef>
#include <type_traits>

/** \addtogroup Xilinx Xilinx vendor extensions
    @{
*/

namespace cl {
namespace sycl {
namespace xilinx {

/** Kind of array partition

    To be used when defining or declaring a vendor-supported partition
    array in kernel.
*/
namespace partition {
  /** Three real partition types: cyclic, block, complete.

      none represents non partitioned standard array.
  */
  enum class type {
    cyclic,
    block,
    complete,
    none
  };


  /** Represent a cyclic partition.

      The single array would be partitioned into several small physical
      memories in this case. These small physical memories can be
      accessed simultaneously which drive the performance. Each
      element in the array would be partitioned to each memory in order
      and cyclically.

      That is if we have a 4-element array which contains 4 integers
      0, 1, 2, and 3. If we set factor to 2, and partition
      dimension to 1 for this cyclic partition array. Then, the
      contents of this array will be distributed to 2 physical
      memories: one contains 1, 3 and the other contains 2,4.

      \param PhyMemNum is the number of physical memories that user wants to
      have.

      \param PDim is the dimension that user wants to apply cyclic partition on.
      If PDim is 0, all dimensions will be partitioned with cyclic order.

      \todo Deal with multi-dimension array. Now, since we can only deal with
      1-dim, PDim is set to 1 by default.
  */
  template <std::size_t PhyMemNum = 1, std::size_t PDim = 1>
  struct cyclic {
    static constexpr auto physical_mem_num = PhyMemNum;
```

```
    static constexpr auto partition_dim = PDim;
    static constexpr auto partition_type = type::cyclic;
  };


  /** Represent a block partition.

      The single array would be partitioned into several small
      physical memories and can be accessed simultaneously, too.
      However, the first physical memory will be filled up first, then
      the next.

      That is if we have a 4-element array which contains 4 integers
      0, 1, 2, and 3. If we set factor to 2, and partition
      dimension to 1 for this cyclic partition array. Then, the
      contents of this array will be distributed to 2 physical
      memories: one contains 1, 2 and the other contains 3,4.

      \param ElmInEachPhyMem is the number of elements in each physical memory
      that user wants to have.

      \param PDim is the dimension that user wants to apply block partition on.
      If PDim is 0, all dimensions will be partitioned with block order.

      \todo Deal with multi-dimension array. Now, since we can only deal with
      1-dim, PDim is set to 1 by default.
  */
  template <std::size_t ElmInEachPhyMem = 1, std::size_t PDim = 1>
  struct block {
    static constexpr auto ele_in_each_physical_mem = ElmInEachPhyMem;
    static constexpr auto partition_dim = PDim;
    static constexpr auto partition_type = type::block;
  };


  /** Represent a complete partition.

      The single array would be partitioned into individual elements.
      That is if we have a 4-element array with one dimension, the
      array is completely partitioned into distributed RAM or 4
      independent registers.

      \param PDim is the dimension that user wants to apply complete partition
      on. If PDim is 0, all dimensions will be completely partitioned.

      \todo Deal with multi-dimension array. Now, since we can only deal with
      1-dim, PDim is set to 1 by default.
  */
  template <std::size_t PDim = 1>
  struct complete {
    static constexpr auto partition_dim = PDim;
    static constexpr auto partition_type = type::complete;
  };


  /** Represent a none partition.

      The single array would be the same as std::array.
  */
  struct none {
    static constexpr auto partition_type = type::none;
  };
}


/** Define an array class with partition feature.

    Since on FPGA, users can customize the memory architecture in the system
    and within the CU. Array partition help us to partition single array to
    multiple memories that can be accessed simultaneously getting higher memory
    bandwidth.

    \param ValueType is the type of element.

    \param Size is the size of the array.

    \param PartitionType is the array partition type: cyclic, block, and
    complete. The default type is none.

    \todo Deal with multi-dimension array.
*/
template <typename ValueType,
          std::size_t Size,
          typename PartitionType = partition::none>
struct partition_array {
  /** Store the array elements.

```

```
      Note that it means default initialization for partition_array
      elements, which is lazily convenient for heterogeneous
      computing */
  ValueType elems[Size];
  /// The number of elements of the 1-D array
  static constexpr auto array_size = Size;
  /// The kind of partitioning
  static constexpr auto partition_type = PartitionType::partition_type;
  /// Type of array elements
  using element_type = ValueType;


  /// Provide iterator
  auto begin() { return elems; }
  const auto begin() const { return elems; }
  auto end() { return elems+Size; }
  const auto end() const { return elems+Size; }


  /// Evaluate size
  constexpr auto size() const noexcept {
    return Size;
  }


  /// Construct an array
  partition_array() {
    // Add the intrinsic according expressing to the target compiler the
    // partitioning to use
    if constexpr (partition_type == partition::type::cyclic)
      _ssdm_SpecArrayPartition(&(*this)[0], PartitionType::partition_dim,
                               "CYCLIC", PartitionType::physical_mem_num, "");
    if constexpr (partition_type == partition::type::block)
      _ssdm_SpecArrayPartition(&(*this)[0], PartitionType::partition_dim,
                               "BLOCK", PartitionType::ele_in_each_physical_mem,
                               "");
    if constexpr (partition_type == partition::type::complete)
      _ssdm_SpecArrayPartition(&(*this)[0], PartitionType::partition_dim,
                               "COMPLETE", 0, "");
  }


  /// A constructor from some container
  template <typename SomeContainer>
  partition_array(const SomeContainer &src)
    : partition_array { } {
    /// \todo Find a way to specialize this with a safer
    /// implementation when the size of src is at least constexpr
    std::copy_n(std::begin(src), Size, begin());
  }


  /// Construct an array from initializer_list
  template <typename SourceBasicType,
            // Only use this constructor for a real element-oriented
            // initializer_list we can assign, not for the case
            // partition_array<> a = { some_other_array }
            typename = std::enable_if_t<std::is_convertible<SourceBasicType,
                                                            ValueType>::value>>
  constexpr partition_array(std::initializer_list<SourceBasicType> l)
    : partition_array { } {
    /// \todo Find a way to specialize this with a safer
    /// implementation when the size of src is at least constexpr
    /// This does not work...
    /// static_assert(l.size() == Size);
    std::copy_n(std::begin(l), Size, begin());
  }


  /// Provide a subscript operator
  constexpr ValueType& operator[](std::size_t i) {
    return elems[i];
  }


  constexpr const ValueType& operator[](std::size_t i) const {
    return elems[i];
  }


  /// Return the partition type of the array
  constexpr auto get_partition_type() const {
    return partition_type;
  }
};

/// @} End the Xilinx Doxygen group
```

```

}
}
}

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif// TRISYCL_SYCL_VENDOR_XILINX_PARTITION_ARRAY_HPP
```

11.165 include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp File Reference

This graph shows which files directly or indirectly include this file:

Macros

- #define _ssdm_op_SpecDataflowPipeline(...) do { } while (0)
- #define _ssdm_op_SpecPipeline(...) do { } while (0)
- #define _ssdm_SpecArrayPartition(...) do { } while (0)

11.166 ssdm_inst.hpp

```
#ifndef TRISYCL_SYCL_VENDOR_XILINX_SSDM_INST_HPP
#define TRISYCL_SYCL_VENDOR_XILINX_SSDM_INST_HPP


/** \file This file defines some of the SSDM intrinsics used in Xilinx
    tools.

    This file is distributed under the University of Illinois Open Source
    License. See LICENSE.TXT for details.
*/

/** \addtogroup Xilinx Xilinx vendor extensions
    @{
*/

#ifdef TRISYCL_DEVICE
extern "C" {
  /// SSDM Intrinsics: dataflow operation
  void _ssdm_op_SpecDataflowPipeline(...)
```

```
    __attribute__ ((nothrow, noinline, weak));
  /// SSDM Intrinsics: pipeline operation
  void _ssdm_op_SpecPipeline(...) __attribute__ ((nothrow, noinline, weak
));
  /// SSDM Intrinsics: array partition operation
  void _ssdm_SpecArrayPartition(...) __attribute__ ((nothrow, noinline
, weak));
}
#else
/* If not on device, just ignore the intrinsics as defining them as
   empty variadic macros replaced by an empty do-while to avoid some
   warning when compiling (and used in an if branch */
#define _ssdm_op_SpecDataflowPipeline(...) do { } while (0)
#define _ssdm_op_SpecPipeline(...) do { } while (0)
#define _ssdm_SpecArrayPartition(...) do { } while (0)
#endif

/// @} End the Xilinx Doxygen group

/*
    # Some Emacs stuff:
    ### Local Variables:
    ### ispell-local-dictionary: "american"
    ### eval: (flyspell-prog-mode)
    ### End:
*/

#endif // TRISYCL_SYCL_VENDOR_XILINX_SSDM_INST_HPP
```

Index

__SYCL_SINGLE_SOURCE__
Manage default configuration and types, 367
__attribute__
cl::sycl::detail, 493
opencl_spir.h, 749, 761
__kernel_exec
opencl_spir.h, 716
_ssdm_SpecArrayPartition
Xilinx vendor extensions, 477
_ssdm_op_SpecDataflowPipeline
Xilinx vendor extensions, 477
_ssdm_op_SpecPipeline
Xilinx vendor extensions, 477
~accessor
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
~buffer
cl::sycl::detail::buffer, 100
~buffer_base
cl::sycl::detail::buffer_base, 113
~buffer_waiter
cl::sycl::detail::buffer_waiter, 124
~context
cl::sycl::detail::context, 237
~device
cl::sycl::detail::device, 254
~device_selector
cl::sycl::device_selector, 272
~error_handler
cl::sycl::error_handler, 378
~opencl_context
cl::sycl::detail::opencl_context, 535
~opencl_device
cl::sycl::detail::opencl_device, 542
~opencl_platform
cl::sycl::detail::opencl_platform, 296
~opencl_queue
cl::sycl::detail::opencl_queue, 556
~pipe_accessor
cl::sycl::detail::pipe_accessor, 163
~pipe_reservation
cl::sycl::detail::pipe_reservation, 178
~platform
cl::sycl::detail::platform, 300
~queue
cl::sycl::detail::queue, 564

a
opencl_spir.h, 772
accelerator
Platforms, contexts, devices and queues, 329
access
cl::sycl::detail::buffer, 109
accessor
cl::sycl::accessor, 53–55
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >, 70
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, 67
cl::sycl::detail::accessor, 77
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
cl::sycl::drt::accessor, 371
accessor_detail
cl::sycl::accessor, 52
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >, 70
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, 67
cl::sycl::pipe_reservation, 187
accessor_type
cl::sycl::detail::pipe_reservation, 176
cl::sycl::pipe_reservation, 187
accessors
cl::sycl::detail::task, 590
add_buffer
cl::sycl::detail::task, 579
add_buffer_to_task
cl::sycl::detail, 494
Data access and storage in SYCL, 202
add_postlude
cl::sycl::detail::task, 579
add_prelude
cl::sycl::detail::task, 580
add_to_task
cl::sycl::detail::buffer_base, 114
addr_space
Dealing with OpenCL address spaces, 227
address_space
cl::sycl::detail::address_space_base, 223
Dealing with OpenCL address spaces, 231
address_space/detail/address_space.hpp
TRISYCL_CONSTANT_AS, 621
TRISYCL_GENERIC_AS, 622
TRISYCL_GLOBAL_AS, 622
TRISYCL_LOCAL_AS, 622
TRISYCL_PRIVATE_AS, 622
address_space_array
cl::sycl::detail::address_space_array, 211

address_space_fundamental
cl::sycl::detail::address_space_fundamental, 215
address_space_object
cl::sycl::detail::address_space_object, 217
address_space_ptr
cl::sycl::detail::address_space_ptr, 221
address_space_variable
cl::sycl::detail::address_space_variable, 225, 226
all
Platforms, contexts, devices and queues, 329
alloc
cl::sycl::detail::buffer, 110
allocate_accessor
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 39
allocate_buffer
cl::sycl::detail::buffer, 101
allocation
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 49
cl::sycl::detail::buffer, 110
allocator_type
cl::sycl::buffer, 128
array
cl::sycl::detail::accessor, 92
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 49
array_size
cl::sycl::xilinx::partition_array, 476
array_type
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 36
array_view_type
cl::sycl::detail::accessor, 74
as_char
opencl_spir.h, 717
as_char16
opencl_spir.h, 717
as_char2
opencl_spir.h, 717
as_char3
opencl_spir.h, 717
as_char4
opencl_spir.h, 717
as_char8
opencl_spir.h, 718
as_double
opencl_spir.h, 718
as_double16
opencl_spir.h, 718
as_double2
opencl_spir.h, 718
as_double3
opencl_spir.h, 718
as_double4
opencl_spir.h, 718
as_double8
opencl_spir.h, 719
as_float
opencl_spir.h, 719
as_float16
opencl_spir.h, 719
as_float2
opencl_spir.h, 719
as_float3
opencl_spir.h, 719
as_float4
opencl_spir.h, 719
as_float8
opencl_spir.h, 720
as_half
opencl_spir.h, 720
as_half16
opencl_spir.h, 720
as_half2
opencl_spir.h, 720
as_half3
opencl_spir.h, 720
as_half4
opencl_spir.h, 720
as_half8
opencl_spir.h, 721
as_int
opencl_spir.h, 721
as_int16
opencl_spir.h, 721
as_int2
opencl_spir.h, 721
as_int3
opencl_spir.h, 721
as_int4
opencl_spir.h, 721
as_int8
opencl_spir.h, 722
as_long
opencl_spir.h, 722
as_long16
opencl_spir.h, 722
as_long2
opencl_spir.h, 722
as_long3
opencl_spir.h, 722
as_long4
opencl_spir.h, 722
as_long8
opencl_spir.h, 723
as_short
opencl_spir.h, 723
as_short16
opencl_spir.h, 723
as_short2
opencl_spir.h, 723
as_short3
opencl_spir.h, 723
as_short4
opencl_spir.h, 723

as_short8
opencl_spir.h, 724
as_uchar
opencl_spir.h, 724
as_uchar16
opencl_spir.h, 724
as_uchar2
opencl_spir.h, 724
as_uchar3
opencl_spir.h, 724
as_uchar4
opencl_spir.h, 724
as_uchar8
opencl_spir.h, 725
as_uint
opencl_spir.h, 725
as_uint16
opencl_spir.h, 725
as_uint2
opencl_spir.h, 725
as_uint3
opencl_spir.h, 725
as_uint4
opencl_spir.h, 725
as_uint8
opencl_spir.h, 726
as_ulong
opencl_spir.h, 726
as_ulong16
opencl_spir.h, 726
as_ulong2
opencl_spir.h, 726
as_ulong3
opencl_spir.h, 726
as_ulong4
opencl_spir.h, 726
as_ulong8
opencl_spir.h, 727
as_ushort
opencl_spir.h, 727
as_ushort16
opencl_spir.h, 727
as_ushort2
opencl_spir.h, 727
as_ushort3
opencl_spir.h, 727
as_ushort4
opencl_spir.h, 727
as_ushort8
opencl_spir.h, 728
assume_validity
cl::sycl::detail::pipe_reservation, 179
async_handler
Error handling, 402
automatic
Platforms, contexts, devices and queues, 329

b
opencl_spir.h, 773
barrier
cl::sycl::nd_item, 428
basic_type
cl::sycl::vec, 466
begin
cl::sycl::accessor, 55
cl::sycl::detail::accessor, 78
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 39
cl::sycl::detail::pipe_reservation, 180
cl::sycl::pipe_reservation, 190
cl::sycl::xilinx::partition_array, 473, 474
binary
cl::sycl::drt::code::program, 562
binary_size
cl::sycl::drt::code::program, 562
blocking
cl::sycl::detail::pipe_accessor, 169
cl::sycl::detail::pipe_reservation, 184
cl::sycl::pipe_reservation, 196
buf
cl::sycl::detail::accessor, 92
buffer
cl::sycl::buffer, 129–135
cl::sycl::detail::buffer, 97–99
cl::sycl::drt::accessor, 372
buffer_add_to_task
Data access and storage in SYCL, 202
buffer_allocator
Data access and storage in SYCL, 201
buffer_base
cl::sycl::detail::buffer_base, 113
buffer_cache
cl::sycl::detail::buffer_base, 121
buffer_waiter
cl::sycl::detail::buffer_waiter, 124
buffers_in_use
cl::sycl::detail::task, 591

c
cl::sycl::detail::cache, 509
cl::sycl::detail::opencl_context, 539
opencl_spir.h, 773
CHAR_BIT
opencl_spir.h, 728
CHAR_MAX
opencl_spir.h, 728
CHAR_MIN
opencl_spir.h, 728
CL_SYCL_LANGUAGE_VERSION
Manage default configuration and types, 367
CLK_ADDRESS_CLAMP_TO_EDGE
opencl_spir.h, 729
CLK_ADDRESS_CLAMP
opencl_spir.h, 728
CLK_ADDRESS_MIRRORED_REPEAT
opencl_spir.h, 729
CLK_ADDRESS_NONE
opencl_spir.h, 729

CLK_ADDRESS_REPEAT
opencl_spir.h, 729
CLK_ARGB
opencl_spir.h, 729
CLK_BGRA
opencl_spir.h, 730
CLK_DEPTH_STENCIL
opencl_spir.h, 730
CLK_DEPTH
opencl_spir.h, 730
CLK_FILTER_LINEAR
opencl_spir.h, 730
CLK_FILTER_NEAREST
opencl_spir.h, 730
CLK_FLOAT
opencl_spir.h, 730
CLK_GLOBAL_MEM_FENCE
opencl_spir.h, 731
CLK_HALF_FLOAT
opencl_spir.h, 731
CLK_INTENSITY
opencl_spir.h, 731
CLK_LOCAL_MEM_FENCE
opencl_spir.h, 731
CLK_LUMINANCE
opencl_spir.h, 731
CLK_NORMALIZED_COORDS_FALSE
opencl_spir.h, 731
CLK_NORMALIZED_COORDS_TRUE
opencl_spir.h, 732
CLK_RGBA
opencl_spir.h, 733
CLK_RGBx
opencl_spir.h, 733
CLK_RGB
opencl_spir.h, 732
CLK_RGx
opencl_spir.h, 733
CLK_RA
opencl_spir.h, 732
CLK_RG
opencl_spir.h, 732
CLK_Rx
opencl_spir.h, 733
CLK_SIGNED_INT16
opencl_spir.h, 733
CLK_SIGNED_INT32
opencl_spir.h, 733
CLK_SIGNED_INT8
opencl_spir.h, 734
CLK_SNORM_INT16
opencl_spir.h, 734
CLK_SNORM_INT8
opencl_spir.h, 734
CLK_UNORM_INT16
opencl_spir.h, 734
CLK_UNORM_INT24
opencl_spir.h, 734
CLK_UNORM_INT8
opencl_spir.h, 734
CLK_UNORM_INT_101010
opencl_spir.h, 735
CLK_UNORM_SHORT_555
opencl_spir.h, 735
CLK_UNORM_SHORT_565
opencl_spir.h, 735
CLK_UNSIGNED_INT16
opencl_spir.h, 735
CLK_UNSIGNED_INT32
opencl_spir.h, 735
CLK_UNSIGNED_INT8
opencl_spir.h, 735
CLK_A
opencl_spir.h, 728
CLK_R
opencl_spir.h, 732
cache
cl::sycl::detail::opencl_context, 540
cl::sycl::detail::opencl_device, 547
cl::sycl::detail::opencl_kernel, 553
cl::sycl::detail::opencl_platform, 299
cl::sycl::detail::opencl_queue, 559
call_update_buffer_state
cl::sycl::detail::buffer, 102, 103
capacity
cl::sycl::detail::pipe, 149
cl::sycl::detail::pipe_accessor, 163
cl::sycl::pipe, 173
cl::sycl::static_pipe, 199
cb
cl::sycl::detail::pipe, 158
cb_mutex
cl::sycl::detail::pipe, 158
cbegin
cl::sycl::accessor, 56
cl::sycl::detail::accessor, 79
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 40
cl::sycl::pipe_reservation, 191
cend
cl::sycl::accessor, 56
cl::sycl::detail::accessor, 80
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 41
cl::sycl::pipe_reservation, 191
cl, 479
cl::sycl, 480
exception_ptr_class, 484
function_class, 484
hash_class, 485
min, 486
mutex_class, 485
shared_ptr_class, 485
string_class, 485
TRISYCL_MATH_WRAP2s, 486
TRISYCL_MATH_WRAP, 486

unique_ptr_class, 485
vector_class, 485
weak_ptr_class, 486
y, 487
z, 487
cl::sycl::access, 488
fence_space, 488
mode, 489
target, 489
cl::sycl::accessor, 50
accessor, 53–55
accessor_detail, 52
begin, 55
cbegin, 56
cend, 56
crbegin, 57
crend, 57
dimensionality, 65
end, 57
get_count, 58
get_pointer, 58
get_range, 58
get_size, 59
implementation_t, 52, 66
operator∗, 59, 60
operator[], 60–64
rbegin, 65
rend, 65
cl::sycl::accessor< DataType, 1, AccessMode, access↩ ::target::blocking_pipe >, 69
accessor, 70
accessor_detail, 70
get_pipe_detail, 70
reserve, 71
cl::sycl::accessor< DataType, 1, AccessMode, access↩ ::target::pipe >, 66
accessor, 67
accessor_detail, 67
get_pipe_detail, 68
reserve, 68
cl::sycl::accessor_error, 387
cl::sycl::async_exception, 384
cl::sycl::buffer, 125
allocator_type, 128
buffer, 129–135
const_reference, 128
get_access, 136, 137
get_count, 138
get_range, 138
get_size, 138
implementation_t, 128, 143
is_cached, 139
is_data_up_to_date, 139
is_read_only, 139
mark_as_written, 140
reference, 128
set_final_data, 140–142
use_count, 142
value_type, 129
cl::sycl::cl_exception, 382
cl_code, 384
cl_exception, 383
get_cl_code, 383
cl::sycl::compile_program_error, 392
cl::sycl::context, 242
context, 245–249
get, 250
get_boost_compute, 250
get_boost_queue, 250
get_devices, 251
get_info, 251
get_platform, 252
implementation_t, 245
is_host, 252
cl::sycl::detail, 490
__attribute__, 493
add_buffer_to_task, 494
prevent_arguments_from_optimization, 495
set_kernel_task_marker, 494
cl::sycl::detail::accessor, 71
accessor, 77
array, 92
array_view_type, 74
begin, 78
buf, 92
cbegin, 79
cend, 80
const_iterator, 75
const_reference, 75
const_reverse_iterator, 75
copy_back_cl_buffer, 80
copy_in_cl_buffer, 80
crbegin, 81
crend, 81
dimensionality, 93
element, 75
end, 81
get_buffer, 82
get_cl_buffer, 82
get_count, 83
get_pointer, 84
get_range, 84
get_size, 84
handler, 93
is_read_access, 85
is_write_access, 86
iterator, 75
operator∗, 86, 87
operator[], 87–89
rbegin, 90
reference, 76
register_accessor, 90
rend, 92
reverse_iterator, 76
value_type, 76
writable_array_view_type, 76

cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 33
~accessor, 38
accessor, 38
allocate_accessor, 39
allocation, 49
array, 49
array_type, 36
begin, 39
cbegin, 40
cend, 41
const_iterator, 36
const_reference, 36
const_reverse_iterator, 37
crbegin, 41
crend, 41
deallocate_accessor, 41
dimensionality, 49
element, 37
end, 42
get_count, 42
get_range, 42
get_size, 43
handler, 50
is_read_access, 43
is_write_access, 44
iterator, 37
operator*, 44, 45
operator[], 45–47
rbegin, 48
reference, 37
rend, 48
reverse_iterator, 37
value_type, 38
writable_array_type, 38
cl::sycl::detail::accessor_base, 31
get_cl_buffer, 32
get_order, 32
order, 33
set_order, 32
task, 33
cl::sycl::detail::address_space_array, 209
address_space_array, 211
super, 211
cl::sycl::detail::address_space_base, 221
address_space, 223
opencl_type, 223
type, 223
cl::sycl::detail::address_space_fundamental, 212
address_space_fundamental, 215
super, 214
cl::sycl::detail::address_space_object, 215
address_space_object, 217
opencl_type, 217
operator opencl_type &, 218
cl::sycl::detail::address_space_ptr, 218
address_space_ptr, 221
pointer_t, 220
reference_t, 221
super, 221
cl::sycl::detail::address_space_variable, 223
address_space_variable, 225, 226
get_address, 226
opencl_type, 225
operator opencl_type &, 226
super, 225
variable, 227
cl::sycl::detail::buffer, 93
~buffer, 100
access, 109
alloc, 110
allocate_buffer, 101
allocation, 110
buffer, 97–99
call_update_buffer_state, 102, 103
copy_if_modified, 110
data_host, 110
deallocate_buffer, 103
detail::accessor, 109
element, 97
final_write_back, 111
get_count, 104
get_destructor_future, 104
get_range, 105
get_size, 106
input_shared_pointer, 111
mark_as_written, 106
modified, 111
non_const_value_type, 97
set_final_data, 107, 108
track_access_mode, 108
value_type, 97
cl::sycl::detail::buffer_base, 111
~buffer_base, 113
add_to_task, 114
buffer_base, 113
buffer_cache, 121
create_in_cache, 114
fresh_ctx, 121
get_cl_buffer, 115
get_latest_producer, 115
is_cached, 116
is_data_up_to_date, 116
latest_producer, 121
latest_producer_mutex, 122
notify_buffer_destructor, 122
number_of_users, 122
ready, 122
ready_mutex, 122
release, 116
set_latest_producer, 117
sync_with_host, 117
update_buffer_state, 118
use, 120
wait, 120
cl::sycl::detail::buffer_waiter, 123

~buffer_waiter, 124
buffer_waiter, 124
implementation_t, 124, 125
cl::sycl::detail::cache
c, 509
get_or_register, 506
key_type, 506
m, 509
remove, 507
value_type, 506
cl::sycl::detail::cache< Key, Value >, 505
cl::sycl::detail::container_element_aspect, 343
const_pointer, 344
const_reference, 344
pointer, 344
reference, 344
value_type, 344
cl::sycl::detail::context, 236
~context, 237
get, 237
get_boost_compute, 237
get_boost_queue, 237
get_devices, 238
get_platform, 238
is_host, 238
cl::sycl::detail::debug, 364
cl::sycl::detail::device, 253
~device, 254
get, 254
get_boost_compute, 254
get_platform, 254
has_extension, 255
is_accelerator, 255
is_cpu, 255
is_gpu, 255
is_host, 255
cl::sycl::detail::display_vector, 364
display, 365
cl::sycl::detail::expand_to_vector, 338
cl::sycl::detail::expand_to_vector< V, Tuple, true >, 338
cl::sycl::detail::host_context, 238
get, 240
get_boost_compute, 240
get_boost_queue, 240
get_devices, 241
get_platform, 241
is_host, 242
cl::sycl::detail::host_device, 523
get, 524
get_boost_compute, 524
get_platform, 525
has_extension, 525
is_accelerator, 526
is_cpu, 526
is_gpu, 527
is_host, 527
cl::sycl::detail::host_platform, 290
get, 292
get_boost_compute, 292
get_info_string, 292
has_extension, 293
is_host, 294
platform_extensions, 294
cl::sycl::detail::host_queue, 528
get, 529
get_boost_compute, 529
get_context, 530
get_device, 530
is_host, 530
cl::sycl::detail::kernel, 284
get, 285
get_boost_compute, 285
single_task, 286
TRISYCL_ParallelForKernel_RANGE, 286
cl::sycl::detail::ocl_type, 207
type, 207
cl::sycl::detail::ocl_type< T, constant_address_space >, 207
type, 207
cl::sycl::detail::ocl_type< T, global_address_space >, 208
type, 208
cl::sycl::detail::ocl_type< T, local_address_space >, 208
type, 208
cl::sycl::detail::ocl_type< T, private_address_space >, 209
type, 209
cl::sycl::detail::opencl_context, 533
~opencl_context, 535
c, 539
cache, 540
get, 535
get_boost_compute, 536
get_boost_queue, 536
get_devices, 536
get_platform, 537
instance, 538
is_host, 539
opencl_context, 535
q, 540
cl::sycl::detail::opencl_device, 541
~opencl_device, 542
cache, 547
d, 547
get, 543
get_boost_compute, 543
get_platform, 544
has_extension, 544
instance, 545
is_accelerator, 546
is_cpu, 546
is_gpu, 546
is_host, 547
opencl_device, 542
cl::sycl::detail::opencl_kernel, 548

cache, 553
get, 550
get_boost_compute, 550
instance, 551
k, 553
opencl_kernel, 550
single_task, 552
TRISYCL_ParallelForKernel_RANGE, 552
cl::sycl::detail::opencl_platform, 294
~opencl_platform, 296
cache, 299
get, 297
get_boost_compute, 297
get_info_string, 297
has_extension, 298
instance, 298
is_host, 299
opencl_platform, 296
p, 299
cl::sycl::detail::opencl_queue, 554
~opencl_queue, 556
cache, 559
get, 557
get_boost_compute, 557
get_context, 557
get_device, 557
instance, 558
is_host, 559
opencl_queue, 556
q, 560
cl::sycl::detail::parallel_OpenMP_for_iterate, 448
parallel_OpenMP_for_iterate, 448
cl::sycl::detail::parallel_for_iterate, 447
parallel_for_iterate, 448
cl::sycl::detail::parallel_for_iterate< 0, Range, Parallel↩ ForFunctor, Id >, 449
parallel_for_iterate, 449
cl::sycl::detail::pipe, 145
capacity, 149
cb, 158
cb_mutex, 158
debug_mode, 158
empty, 149
empty_with_lock, 149
full, 150
full_with_lock, 150
implementation_t, 148
move_read_reservation_forward, 150
move_write_reservation_forward, 151
pipe, 148
r_rid_q, 158
read, 152
read_done, 158
read_reserved_frozen, 159
reserve_read, 153
reserve_write, 154
reserved_for_reading, 155
reserved_for_writing, 155
rid_iterator, 148
size, 156
size_with_lock, 156
used_for_reading, 159
used_for_writing, 159
value_type, 148
w_rid_q, 159
write, 157
write_done, 159
cl::sycl::detail::pipe_accessor, 160
~pipe_accessor, 163
blocking, 169
capacity, 163
const_reference, 162
empty, 163
full, 164
get_pipe_detail, 164
implementation, 169
mode, 169
ok, 170
operator bool, 164
operator<<, 165
operator>>, 165
pipe_accessor, 162
rank, 170
read, 166
reference, 162
reserve, 167
set_debug, 167
size, 168
target, 170
value_type, 162
write, 168
cl::sycl::detail::pipe_reservation, 174
~pipe_reservation, 178
accessor_type, 176
assume_validity, 179
begin, 180
blocking, 184
commit, 180
const_iterator, 176
end, 181
iterator, 176
mode, 184
ok, 184
operator bool, 182
operator[], 182
p, 184
pipe_reservation, 177, 178
reference, 176
rid, 184
size, 183
target, 185
value_type, 176
cl::sycl::detail::platform, 300
~platform, 300
get, 301
get_boost_compute, 301

get_devices, 301
get_info_string, 301
has_extension, 302
is_host, 302
cl::sycl::detail::queue, 563
~queue, 564
finished, 569
finished_mutex, 569
get, 565
get_boost_compute, 565
get_context, 566
get_device, 566
is_host, 567
kernel_end, 567
kernel_start, 568
queue, 564
running_kernels, 569
wait_for_kernel_execution, 568
cl::sycl::detail::reserve_id, 143
ready, 145
reserve_id, 144
size, 145
start, 145
cl::sycl::detail::shared_ptr_implementation
hash, 573
implementation, 574
operator<, 573
operator==, 574
shared_ptr_implementation, 572
cl::sycl::detail::shared_ptr_implementation< Parent, Implementation >, 570
cl::sycl::detail::singleton
instance, 575
cl::sycl::detail::singleton< T >, 575
cl::sycl::detail::small_array, 344
dimension, 352
dimensionality, 352
element_type, 347
get, 349
operator FinalType, 349
small_array, 347, 348
x, 349
y, 350
z, 351
cl::sycl::detail::small_array_123, 352
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 353
operator BasicType, 355
small_array_123, 354, 355
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >, 355
small_array_123, 356, 357
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >, 358
small_array_123, 359, 360
cl::sycl::detail::task, 576
accessors, 590
add_buffer, 579
add_postlude, 579
add_prelude, 580
buffers_in_use, 591
epilogues, 591
execution_ended, 591
get_compute_buffer, 580
get_kernel, 581
get_queue, 581
kernel, 591
notify_consumers, 582
owner_queue, 592
postlude, 583
prelude, 583
producer_tasks, 592
prologues, 592
ready, 592
ready_mutex, 592
register_accessor, 584
release_buffers, 585
schedule, 585
set_arg, 587, 588
set_kernel, 589
task, 579
wait, 589
wait_for_producers, 590
cl::sycl::device, 256
device, 258, 259
get, 260
get_boost_compute, 261
get_info, 261, 262
get_platform, 262
has_extension, 263
implementation_t, 258
is_accelerator, 263
is_cpu, 264
is_gpu, 264
is_host, 265
type, 265
cl::sycl::device_error, 391
cl::sycl::device_selector, 271
~device_selector, 272
operator(), 273
select_device, 273
cl::sycl::device_type_selector, 266
default_device, 269
device_type, 269
device_type_selector, 268
operator(), 268
cl::sycl::device_typename_selector, 270
device_typename_selector, 271
cl::sycl::drt, 496
cl::sycl::drt::accessor, 369
accessor, 371
buffer, 372
operator[], 371
size, 372
cl::sycl::drt::code, 496
cl::sycl::drt::code::program, 561

binary, 562
binary_size, 562
p, 562
program, 561
cl::sycl::error_handler, 377
~error_handler, 378
default_handler, 378
report_error, 378
cl::sycl::event, 515
event, 515
cl::sycl::event_error, 389
cl::sycl::exception, 379
exception, 381
message, 382
what, 381
cl::sycl::exception_list, 378
cl::sycl::feature_not_supported, 400
cl::sycl::group, 404
dimensionality, 415
get_global_range, 407
get_group_range, 407
get_id, 408
get_linear, 409
get_local_range, 410
get_nd_range, 411
get_offset, 411, 412
group, 406
group_id, 415
ndr, 415
operator[], 412
parallel_for_work_item, 413, 414
cl::sycl::handler, 273
Dimensions, 284
dispatch_set_arg, 276
handler, 275
parallel_for, 277
parallel_for_work_group, 277, 278
schedule_kernel, 279
set_arg, 279–281
set_args, 281
single_task, 282
TRISYCL_parallel_for_functor_GLOBAL, 283
task, 284
cl::sycl::id, 415
id, 416
cl::sycl::image, 143
cl::sycl::info, 497
queue, 498
queue_profiling, 498
cl::sycl::info::param_traits< T, Param >, 560
cl::sycl::invalid_object_error, 395
cl::sycl::invalid_parameter_error, 390
cl::sycl::is_wrapper< T >, 531
cl::sycl::item, 417
dimensionality, 424
display, 419
get_id, 419, 420
get_linear_id, 421
get_offset, 422
get_range, 422
global_index, 424
global_range, 424
item, 418
offset, 424
operator[], 423
set, 423
cl::sycl::kernel, 286
get, 289
handler, 290
implementation_t, 288, 290
kernel, 288, 289
cl::sycl::kernel_error, 386
cl::sycl::link_program_error, 393
cl::sycl::memory_allocation_error, 396
cl::sycl::nd_item, 424
barrier, 428
dimensionality, 442
get_global, 428, 429
get_global_linear_id, 430
get_global_range, 431
get_group, 432, 433
get_group_linear_id, 433
get_item, 434
get_local, 435
get_local_linear_id, 436
get_local_range, 437
get_nd_range, 438
get_num_groups, 439, 440
get_offset, 440
global_index, 442
local_index, 442
ND_range, 442
nd_item, 426, 427
set_global, 441
set_local, 441
cl::sycl::nd_range, 442
dimensionality, 446
display, 444
get_global, 444
get_group, 445
get_local, 445
get_offset, 446
global_range, 447
local_range, 447
nd_range, 444
offset, 447
cl::sycl::nd_range_error, 388
cl::sycl::non_cl_error, 401
cl::sycl::pipe, 170
capacity, 173
get_access, 173
implementation_t, 172, 173
pipe, 172
value_type, 172
cl::sycl::pipe_error, 397
cl::sycl::pipe_reservation, 185

accessor_detail, 187
accessor_type, 187
begin, 190
blocking, 196
cbegin, 191
cend, 191
commit, 192
const_iterator, 187
const_pointer, 187
const_reference, 187
const_reverse_iterator, 188
crbegin, 192
crend, 193
difference_type, 188
end, 193
implementation, 197
iterator, 188
operator bool, 194
operator[], 194
pipe_reservation, 189, 190
pointer, 188
rbegin, 195
reference, 188
rend, 195
reverse_iterator, 188
size, 196
size_type, 189
value_type, 189
cl::sycl::platform, 302
get, 306
get_boost_compute, 306
get_devices, 307
get_info, 307, 308
get_platforms, 308
has_extension, 308
is_host, 309
platform, 304, 305
shared_ptr_implementation, 309
cl::sycl::platform_error, 398
cl::sycl::profiling_error, 399
cl::sycl::queue, 309
get, 317
get_boost_compute, 317
get_context, 317
get_device, 317
get_info, 318
implementation_t, 311, 321
is_host, 318
queue, 312–316
submit, 318, 319
throw_asynchronous, 319
wait, 320
wait_and_throw, 320
cl::sycl::range, 450
get_count, 451
cl::sycl::runtime_error, 385
cl::sycl::static_pipe, 197
capacity, 199
get_access, 200
implementation_t, 199, 201
static_pipe, 199
value_type, 199
cl::sycl::trisycl, 499
cl::sycl::trisycl::default_error_handler, 514
report_error, 515
cl::sycl::vec, 464
basic_type, 466
flatten, 467
flatten_to_tuple, 468
vec, 466, 467
cl::sycl::xilinx, 499
dataflow, 500
pipeline, 500
cl::sycl::xilinx::partition, 501
type, 502
cl::sycl::xilinx::partition::block
ele_in_each_physical_mem, 504
partition_dim, 504
partition_type, 504
cl::sycl::xilinx::partition::block< ElmInEachPhyMem, PDim >, 503
cl::sycl::xilinx::partition::complete
partition_dim, 511
partition_type, 511
cl::sycl::xilinx::partition::complete< PDim >, 510
cl::sycl::xilinx::partition::cyclic
partition_dim, 513
partition_type, 513
physical_mem_num, 513
cl::sycl::xilinx::partition::cyclic< PhyMemNum, PDim >, 512
cl::sycl::xilinx::partition::none, 532
partition_type, 533
cl::sycl::xilinx::partition_array, 470
array_size, 476
begin, 473, 474
element_type, 472
elems, 476
end, 474
get_partition_type, 474
operator[], 475
partition_array, 472, 473
partition_type, 476
size, 475
cl_code
cl::sycl::cl_exception, 384
cl_exception
cl::sycl::cl_exception, 383
cl_mem_fence_flags
opencl_spir.h, 760
cmp
opencl_spir.h, 773
color
opencl_spir.h, 773
commit
cl::sycl::detail::pipe_reservation, 180

cl::sycl::pipe_reservation, 192
const_func
opencl_spir.h, 736
const_iterator
cl::sycl::detail::accessor, 75
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 36
cl::sycl::detail::pipe_reservation, 176
cl::sycl::pipe_reservation, 187
const_pointer
cl::sycl::detail::container_element_aspect, 344
cl::sycl::pipe_reservation, 187
const_reference
cl::sycl::buffer, 128
cl::sycl::detail::accessor, 75
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 36
cl::sycl::detail::container_element_aspect, 344
cl::sycl::detail::pipe_accessor, 162
cl::sycl::pipe_reservation, 187
const_reverse_iterator
cl::sycl::detail::accessor, 75
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 37
cl::sycl::pipe_reservation, 188
constant
Dealing with OpenCL address spaces, 227
constant_ptr
Dealing with OpenCL address spaces, 228
context
cl::sycl::context, 245–249
Platforms, contexts, devices and queues, 323
coord
opencl_spir.h, 773
copy_back_cl_buffer
cl::sycl::detail::accessor, 80
copy_if_modified
cl::sycl::detail::buffer, 110
copy_in_cl_buffer
cl::sycl::detail::accessor, 80
cosval
opencl_spir.h, 773
cpu
Platforms, contexts, devices and queues, 329
cpu_selector
Platforms, contexts, devices and queues, 321
crbegin
cl::sycl::accessor, 57
cl::sycl::detail::accessor, 81
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 41
cl::sycl::pipe_reservation, 192
create_in_cache
cl::sycl::detail::buffer_base, 114
crend
cl::sycl::accessor, 57
cl::sycl::detail::accessor, 81
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 41
cl::sycl::pipe_reservation, 193
custom
Platforms, contexts, devices and queues, 329

d
cl::sycl::detail::opencl_device, 547
DBL_DIG
opencl_spir.h, 736
DBL_EPSILON
opencl_spir.h, 736
DBL_MANT_DIG
opencl_spir.h, 737
DBL_MAX_10_EXP
opencl_spir.h, 737
DBL_MAX_EXP
opencl_spir.h, 738
DBL_MAX
opencl_spir.h, 737
DBL_MIN_10_EXP
opencl_spir.h, 738
DBL_MIN_EXP
opencl_spir.h, 739
DBL_MIN
opencl_spir.h, 738
DBL_RADIX
opencl_spir.h, 739
Data access and storage in SYCL, 29
add_buffer_to_task, 202
buffer_add_to_task, 202
buffer_allocator, 201
get_pipe_detail, 203
image_allocator, 201
map_allocator, 202
waiter, 204
data_host
cl::sycl::detail::buffer, 110
dataflow
cl::sycl::xilinx, 500
Dealing with OpenCL address spaces, 205
addr_space, 227
address_space, 231
constant, 227
constant_ptr, 228
generic, 228
global, 228
global_ptr, 229
local, 229
local_ptr, 229
make_multi, 231
multi_ptr, 230
priv, 230
private_ptr, 230
deallocate_accessor
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 41
deallocate_buffer
cl::sycl::detail::buffer, 103

debug.hpp
TRISYCL_DUMP_T, 687
TRISYCL_DUMP, 686
TRISYCL_INTERNAL_DUMP, 687
debug_mode
cl::sycl::detail::pipe, 158
Debugging and tracing support, 364
trace_kernel, 365
default_device
cl::sycl::device_type_selector, 269
default_handler
cl::sycl::error_handler, 378
default_selector
Platforms, contexts, devices and queues, 322
denorm
Platforms, contexts, devices and queues, 329
detail::accessor
cl::sycl::detail::buffer, 109
device
cl::sycl::device, 258, 259
Platforms, contexts, devices and queues, 324
Device-side runtime implementation, 369
kernel_IR, 375
serialize_accessor_arg, 372
serialize_arg, 373
set_kernel, 374
device::get_info< info::device::device_type >
Platforms, contexts, devices and queues, 332
device::get_info< info::device::local_mem_size >
Platforms, contexts, devices and queues, 332
device::get_info< info::device::max_compute_units >
Platforms, contexts, devices and queues, 332
device::get_info< info::device::max_mem_alloc_size >
Platforms, contexts, devices and queues, 333
device::get_info< info::device::max_work_group_size >
Platforms, contexts, devices and queues, 333
device::get_info< info::device::name >
Platforms, contexts, devices and queues, 333
device::get_info< info::device::profile >
Platforms, contexts, devices and queues, 334
device::get_info< info::device::vendor >
Platforms, contexts, devices and queues, 334
device_affinity_domain
Platforms, contexts, devices and queues, 327
device_exec_capabilities
Platforms, contexts, devices and queues, 322
device_execution_capabilities
Platforms, contexts, devices and queues, 327
device_fp_config
Platforms, contexts, devices and queues, 322
device_partition_property
Platforms, contexts, devices and queues, 327
device_partition_type
Platforms, contexts, devices and queues, 328
device_queue_properties
Platforms, contexts, devices and queues, 323
device_type
cl::sycl::device_type_selector, 269
Platforms, contexts, devices and queues, 328
device_type_selector
cl::sycl::device_type_selector, 268
device_typename_selector
cl::sycl::device_typename_selector, 271
devices
Platforms, contexts, devices and queues, 324
difference_type
cl::sycl::pipe_reservation, 188
dimension
cl::sycl::detail::small_array, 352
dimensionality
cl::sycl::accessor, 65
cl::sycl::detail::accessor, 93
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 49
cl::sycl::detail::small_array, 352
cl::sycl::group, 415
cl::sycl::item, 424
cl::sycl::nd_item, 442
cl::sycl::nd_range, 446
Dimensions
cl::sycl::handler, 284
dispatch_set_arg
cl::sycl::handler, 276
display
cl::sycl::detail::display_vector, 365
cl::sycl::item, 419
cl::sycl::nd_range, 444
dst_stride
opencl_spir.h, 774

edge1
opencl_spir.h, 774
ele_in_each_physical_mem
cl::sycl::xilinx::partition::block, 504
element
cl::sycl::detail::accessor, 75
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 37
cl::sycl::detail::buffer, 97
element_type
cl::sycl::detail::small_array, 347
cl::sycl::xilinx::partition_array, 472
elems
cl::sycl::xilinx::partition_array, 476
empty
cl::sycl::detail::pipe, 149
cl::sycl::detail::pipe_accessor, 163
empty_with_lock
cl::sycl::detail::pipe, 149
end
cl::sycl::accessor, 57
cl::sycl::detail::accessor, 81
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 42
cl::sycl::detail::pipe_reservation, 181
cl::sycl::pipe_reservation, 193
cl::sycl::xilinx::partition_array, 474

epilogues
cl::sycl::detail::task, 591
Error handling, 376
async_handler, 402
exception_ptr, 402
event
cl::sycl::event, 515
opencl_spir.h, 774
event_list
opencl_spir.h, 774
exception
cl::sycl::exception, 381
exception_ptr
Error handling, 402
exception_ptr_class
cl::sycl, 484
execution_ended
cl::sycl::detail::task, 591
exp
opencl_spir.h, 774
expand
Helpers to do array and tuple conversion, 339, 340
Expressing parallelism through kernels, 403
make_id, 452, 453
make_range, 453, 454
parallel_for, 454–457
parallel_for_global_offset, 458
parallel_for_work_item, 459
parallel_for_workgroup, 460
parallel_for_workitem, 461
extensions
Platforms, contexts, devices and queues, 325

FLT_DIG
opencl_spir.h, 739
FLT_EPSILON
opencl_spir.h, 739
FLT_MANT_DIG
opencl_spir.h, 739
FLT_MAX_10_EXP
opencl_spir.h, 740
FLT_MAX_EXP
opencl_spir.h, 740
FLT_MAX
opencl_spir.h, 740
FLT_MIN_10_EXP
opencl_spir.h, 740
FLT_MIN_EXP
opencl_spir.h, 740
FLT_MIN
opencl_spir.h, 740
FLT_RADIX
opencl_spir.h, 741
FP_ILOGB0
opencl_spir.h, 741
FP_ILOGBNAN
opencl_spir.h, 741
fence_space
cl::sycl::access, 488
fill_tuple
Helpers to do array and tuple conversion, 340
final_write_back
cl::sycl::detail::buffer, 111
finished
cl::sycl::detail::queue, 569
finished_mutex
cl::sycl::detail::queue, 569
flatten
cl::sycl::vec, 467
flatten_to_tuple
cl::sycl::vec, 468
fma
Platforms, contexts, devices and queues, 330
fp_config
Platforms, contexts, devices and queues, 329
fresh_ctx
cl::sycl::detail::buffer_base, 121
full
cl::sycl::detail::pipe, 150
cl::sycl::detail::pipe_accessor, 164
full_with_lock
cl::sycl::detail::pipe, 150
function_class
cl::sycl, 484

generic
Dealing with OpenCL address spaces, 228
get
cl::sycl::context, 250
cl::sycl::detail::context, 237
cl::sycl::detail::device, 254
cl::sycl::detail::host_context, 240
cl::sycl::detail::host_device, 524
cl::sycl::detail::host_platform, 292
cl::sycl::detail::host_queue, 529
cl::sycl::detail::kernel, 285
cl::sycl::detail::opencl_context, 535
cl::sycl::detail::opencl_device, 543
cl::sycl::detail::opencl_kernel, 550
cl::sycl::detail::opencl_platform, 297
cl::sycl::detail::opencl_queue, 557
cl::sycl::detail::platform, 301
cl::sycl::detail::queue, 565
cl::sycl::detail::small_array, 349
cl::sycl::device, 260
cl::sycl::kernel, 289
cl::sycl::platform, 306
cl::sycl::queue, 317
get_access
cl::sycl::buffer, 136, 137
cl::sycl::pipe, 173
cl::sycl::static_pipe, 200
get_address
cl::sycl::detail::address_space_variable, 226
get_boost_compute
cl::sycl::context, 250
cl::sycl::detail::context, 237
cl::sycl::detail::device, 254

cl::sycl::detail::host_context, 240
cl::sycl::detail::host_device, 524
cl::sycl::detail::host_platform, 292
cl::sycl::detail::host_queue, 529
cl::sycl::detail::kernel, 285
cl::sycl::detail::opencl_context, 536
cl::sycl::detail::opencl_device, 543
cl::sycl::detail::opencl_kernel, 550
cl::sycl::detail::opencl_platform, 297
cl::sycl::detail::opencl_queue, 557
cl::sycl::detail::platform, 301
cl::sycl::detail::queue, 565
cl::sycl::device, 261
cl::sycl::platform, 306
cl::sycl::queue, 317
get_boost_queue
cl::sycl::context, 250
cl::sycl::detail::context, 237
cl::sycl::detail::host_context, 240
cl::sycl::detail::opencl_context, 536
get_buffer
cl::sycl::detail::accessor, 82
get_cl_buffer
cl::sycl::detail::accessor, 82
cl::sycl::detail::accessor_base, 32
cl::sycl::detail::buffer_base, 115
get_cl_code
cl::sycl::cl_exception, 383
get_compute_buffer
cl::sycl::detail::task, 580
get_context
cl::sycl::detail::host_queue, 530
cl::sycl::detail::opencl_queue, 557
cl::sycl::detail::queue, 566
cl::sycl::queue, 317
get_count
cl::sycl::accessor, 58
cl::sycl::buffer, 138
cl::sycl::detail::accessor, 83
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 42
cl::sycl::detail::buffer, 104
cl::sycl::range, 451
get_destructor_future
cl::sycl::detail::buffer, 104
get_device
cl::sycl::detail::host_queue, 530
cl::sycl::detail::opencl_queue, 557
cl::sycl::detail::queue, 566
cl::sycl::queue, 317
get_devices
cl::sycl::context, 251
cl::sycl::detail::context, 238
cl::sycl::detail::host_context, 241
cl::sycl::detail::opencl_context, 536
cl::sycl::detail::platform, 301
cl::sycl::platform, 307
Platforms, contexts, devices and queues, 334–336
get_global
cl::sycl::nd_item, 428, 429
cl::sycl::nd_range, 444
get_global_linear_id
cl::sycl::nd_item, 430
get_global_range
cl::sycl::group, 407
cl::sycl::nd_item, 431
get_group
cl::sycl::nd_item, 432, 433
cl::sycl::nd_range, 445
get_group_linear_id
cl::sycl::nd_item, 433
get_group_range
cl::sycl::group, 407
get_id
cl::sycl::group, 408
cl::sycl::item, 419, 420
get_info
cl::sycl::context, 251
cl::sycl::device, 261, 262
cl::sycl::platform, 307, 308
cl::sycl::queue, 318
get_info_string
cl::sycl::detail::host_platform, 292
cl::sycl::detail::opencl_platform, 297
cl::sycl::detail::platform, 301
get_item
cl::sycl::nd_item, 434
get_kernel
cl::sycl::detail::task, 581
get_latest_producer
cl::sycl::detail::buffer_base, 115
get_linear
cl::sycl::group, 409
get_linear_id
cl::sycl::item, 421
get_local
cl::sycl::nd_item, 435
cl::sycl::nd_range, 445
get_local_linear_id
cl::sycl::nd_item, 436
get_local_range
cl::sycl::group, 410
cl::sycl::nd_item, 437
get_nd_range
cl::sycl::group, 411
cl::sycl::nd_item, 438
get_num_groups
cl::sycl::nd_item, 439, 440
get_offset
cl::sycl::group, 411, 412
cl::sycl::item, 422
cl::sycl::nd_item, 440
cl::sycl::nd_range, 446
get_or_register
cl::sycl::detail::cache, 506
get_order

cl::sycl::detail::accessor_base, 32
get_partition_type
cl::sycl::xilinx::partition_array, 474
get_pipe_detail
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >, 70
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, 68
cl::sycl::detail::pipe_accessor, 164
Data access and storage in SYCL, 203
get_platform
cl::sycl::context, 252
cl::sycl::detail::context, 238
cl::sycl::detail::device, 254
cl::sycl::detail::host_context, 241
cl::sycl::detail::host_device, 525
cl::sycl::detail::opencl_context, 537
cl::sycl::detail::opencl_device, 544
cl::sycl::device, 262
get_platforms
cl::sycl::platform, 308
get_pointer
cl::sycl::accessor, 58
cl::sycl::detail::accessor, 84
get_queue
cl::sycl::detail::task, 581
get_range
cl::sycl::accessor, 58
cl::sycl::buffer, 138
cl::sycl::detail::accessor, 84
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 42
cl::sycl::detail::buffer, 105
cl::sycl::item, 422
get_size
cl::sycl::accessor, 59
cl::sycl::buffer, 138
cl::sycl::detail::accessor, 84
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 43
cl::sycl::detail::buffer, 106
global
Dealing with OpenCL address spaces, 228
Platforms, contexts, devices and queues, 331
global_config.hpp
TRISYCL_WEAK_ATTRIB_PREFIX, 695
TRISYCL_WEAK_ATTRIB_SUFFIX, 695
global_index
cl::sycl::item, 424
cl::sycl::nd_item, 442
global_mem_cache_type
Platforms, contexts, devices and queues, 330
global_ptr
Dealing with OpenCL address spaces, 229
global_range
cl::sycl::item, 424
cl::sycl::nd_range, 447
gpu
Platforms, contexts, devices and queues, 329
gpu_selector
Platforms, contexts, devices and queues, 323
group
cl::sycl::group, 406
group_id
cl::sycl::group, 415

HUGE_VALF
opencl_spir.h, 741
HUGE_VAL
opencl_spir.h, 741
handler
cl::sycl::detail::accessor, 93
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 50
cl::sycl::handler, 275
cl::sycl::kernel, 290
handler.hpp
TRISYCL_ParallelForFunctor_GLOBAL_OFFSET, 1029
TRISYCL_ParallelForKernel_RANGE, 1030
TRISYCL_parallel_for_functor_GLOBAL, 1029
handler_event, 516
has_extension
cl::sycl::detail::device, 255
cl::sycl::detail::host_device, 525
cl::sycl::detail::host_platform, 293
cl::sycl::detail::opencl_device, 544
cl::sycl::detail::opencl_platform, 298
cl::sycl::detail::platform, 302
cl::sycl::device, 263
cl::sycl::platform, 308
hash
cl::sycl::detail::shared_ptr_implementation, 573
hash_class
cl::sycl, 485
Helpers to do array and tuple conversion, 338
expand, 339, 340
fill_tuple, 340
tuple_to_array, 341
tuple_to_array_iterate, 341
host
Platforms, contexts, devices and queues, 329
host_selector
Platforms, contexts, devices and queues, 323

¡
opencl_spir.h, 774
INFINITY
opencl_spir.h, 742
INT_MAX
opencl_spir.h, 742
INT_MIN
opencl_spir.h, 742
id
cl::sycl::id, 416
image_allocator
Data access and storage in SYCL, 201

implementation
cl::sycl::detail::pipe_accessor, 169
cl::sycl::detail::shared_ptr_implementation, 574
cl::sycl::pipe_reservation, 197
implementation_t
cl::sycl::accessor, 52, 66
cl::sycl::buffer, 128, 143
cl::sycl::context, 245
cl::sycl::detail::buffer_waiter, 124, 125
cl::sycl::detail::pipe, 148
cl::sycl::device, 258
cl::sycl::kernel, 288, 290
cl::sycl::pipe, 172, 173
cl::sycl::queue, 311, 321
cl::sycl::static_pipe, 199, 201
include/CL/sycl.hpp, 593, 594
include/CL/sycl/access.hpp, 595, 596
include/CL/sycl/accessor.hpp, 597, 598
include/CL/sycl/accessor/detail/accessor_base.hpp, 611, 612
include/CL/sycl/accessor/detail/local_accessor.hpp, 613, 615
include/CL/sycl/address_space.hpp, 627, 630
include/CL/sycl/address_space/detail/address_space.hpp, 619, 623
include/CL/sycl/allocator.hpp, 631, 633
include/CL/sycl/buffer.hpp, 640, 641
include/CL/sycl/buffer/detail/accessor.hpp, 604, 605
include/CL/sycl/buffer/detail/buffer.hpp, 633, 635
include/CL/sycl/buffer/detail/buffer_base.hpp, 648, 649
include/CL/sycl/buffer/detail/buffer_waiter.hpp, 653, 655
include/CL/sycl/buffer_allocator.hpp, 656, 657
include/CL/sycl/command_group/detail/task.hpp, 658, 659
include/CL/sycl/context.hpp, 665, 667
include/CL/sycl/context/detail/context.hpp, 663, 664
include/CL/sycl/context/detail/host_context.hpp, 671, 673
include/CL/sycl/context/detail/opencl_context.hpp, 674, 676
include/CL/sycl/detail/SPIR/opencl_spir.h, 711, 778
include/CL/sycl/detail/array_tuple_helpers.hpp, 677, 679
include/CL/sycl/detail/cache.hpp, 681, 682
include/CL/sycl/detail/container_element_aspect.hpp, 684
include/CL/sycl/detail/debug.hpp, 685, 688
include/CL/sycl/detail/default_classes.hpp, 690, 692
include/CL/sycl/detail/global_config.hpp, 693, 696
include/CL/sycl/detail/instantiate_kernel.hpp, 697, 698
include/CL/sycl/detail/linear_id.hpp, 700, 701
include/CL/sycl/detail/shared_ptr_implementation.hpp, 701, 703
include/CL/sycl/detail/singleton.hpp, 704, 705
include/CL/sycl/detail/small_array.hpp, 706, 708
include/CL/sycl/detail/unimplemented.hpp, 982, 983
include/CL/sycl/device.hpp, 987, 988
include/CL/sycl/device/detail/device.hpp, 984, 985
include/CL/sycl/device/detail/device_tail.hpp, 997, 998
include/CL/sycl/device/detail/host_device.hpp, 998, 1000
include/CL/sycl/device/detail/opencl_device.hpp, 1001, 1002
include/CL/sycl/device_runtime.hpp, 1004, 1006
include/CL/sycl/device_selector.hpp, 1009, 1010
include/CL/sycl/device_selector/detail/device_selector_tail.hpp, 1011, 1013
include/CL/sycl/error_handler.hpp, 1015, 1016
include/CL/sycl/event.hpp, 1017, 1018
include/CL/sycl/exception.hpp, 1018, 1021
include/CL/sycl/group.hpp, 1023, 1025
include/CL/sycl/handler.hpp, 1027, 1030
include/CL/sycl/handler_event.hpp, 1037
include/CL/sycl/id.hpp, 1038, 1039
include/CL/sycl/image.hpp, 1040, 1041
include/CL/sycl/info/context.hpp, 669, 671
include/CL/sycl/info/device.hpp, 992, 995
include/CL/sycl/info/param_traits.hpp, 1042, 1044
include/CL/sycl/info/platform.hpp, 1044, 1046
include/CL/sycl/item.hpp, 1053, 1055
include/CL/sycl/kernel.hpp, 1059, 1061
include/CL/sycl/kernel/detail/kernel.hpp, 1056, 1058
include/CL/sycl/kernel/detail/opencl_kernel.hpp, 1062, 1065
include/CL/sycl/math.hpp, 1066, 1070
include/CL/sycl/nd_item.hpp, 1072, 1074
include/CL/sycl/nd_range.hpp, 1077, 1078
include/CL/sycl/opencl_types.hpp, 1079, 1085
include/CL/sycl/parallelism.hpp, 1094, 1096
include/CL/sycl/parallelism/detail/parallelism.hpp, 1087, 1090
include/CL/sycl/pipe.hpp, 1103, 1104
include/CL/sycl/pipe/detail/pipe.hpp, 1096, 1098
include/CL/sycl/pipe/detail/pipe_accessor.hpp, 1106, 1107
include/CL/sycl/pipe_reservation.hpp, 1114, 1115
include/CL/sycl/pipe_reservation/detail/pipe_reservation.hpp, 1110, 1112
include/CL/sycl/platform.hpp, 1050, 1051
include/CL/sycl/platform/detail/host_platform.hpp, 1117, 1119
include/CL/sycl/platform/detail/host_platform_tail.hpp, 1121
include/CL/sycl/platform/detail/opencl_platform.hpp, 1122, 1123
include/CL/sycl/platform/detail/opencl_platform_tail.hpp, 1125, 1126
include/CL/sycl/platform/detail/platform.hpp, 1047, 1049
include/CL/sycl/queue.hpp, 1134, 1136
include/CL/sycl/queue/detail/host_queue.hpp, 1126, 1128
include/CL/sycl/queue/detail/opencl_queue.hpp, 1129, 1130
include/CL/sycl/queue/detail/queue.hpp, 1131, 1133
include/CL/sycl/range.hpp, 1140, 1142
include/CL/sycl/static_pipe.hpp, 1143, 1144

include/CL/sycl/vec.hpp, 1145, 1147
include/CL/sycl/vendor/Xilinx/opt_decorate_func.hpp, 1150
include/CL/sycl/vendor/Xilinx/partition_array.hpp, 1151, 1153
include/CL/sycl/vendor/Xilinx/ssdm_inst.hpp, 1156
input_shared_pointer
    cl::sycl::detail::buffer, 111
instance
    cl::sycl::detail::opencl_context, 538
    cl::sycl::detail::opencl_device, 545
    cl::sycl::detail::opencl_kernel, 551
    cl::sycl::detail::opencl_platform, 298
    cl::sycl::detail::opencl_queue, 558
    cl::sycl::detail::singleton, 575
intptr_t
    opencl_spir.h, 760
iptr
    opencl_spir.h, 775
is_accelerator
    cl::sycl::detail::device, 255
    cl::sycl::detail::host_device, 526
    cl::sycl::detail::opencl_device, 546
    cl::sycl::device, 263
is_cached
    cl::sycl::buffer, 139
    cl::sycl::detail::buffer_base, 116
is_cpu
    cl::sycl::detail::device, 255
    cl::sycl::detail::host_device, 526
    cl::sycl::detail::opencl_device, 546
    cl::sycl::device, 264
is_data_up_to_date
    cl::sycl::buffer, 139
    cl::sycl::detail::buffer_base, 116
is_gpu
    cl::sycl::detail::device, 255
    cl::sycl::detail::host_device, 527
    cl::sycl::detail::opencl_device, 546
    cl::sycl::device, 264
is_host
    cl::sycl::context, 252
    cl::sycl::detail::context, 238
    cl::sycl::detail::device, 255
    cl::sycl::detail::host_context, 242
    cl::sycl::detail::host_device, 527
    cl::sycl::detail::host_platform, 294
    cl::sycl::detail::host_queue, 530
    cl::sycl::detail::opencl_context, 539
    cl::sycl::detail::opencl_device, 547
    cl::sycl::detail::opencl_platform, 299
    cl::sycl::detail::opencl_queue, 559
    cl::sycl::detail::platform, 302
    cl::sycl::detail::queue, 567
    cl::sycl::device, 265
    cl::sycl::platform, 309
    cl::sycl::queue, 318
is_read_access
    cl::sycl::detail::accessor, 85
    cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 43
is_read_only
    cl::sycl::buffer, 139
is_write_access
    cl::sycl::detail::accessor, 86
    cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 44
item
    cl::sycl::item, 418
iterator
    cl::sycl::detail::accessor, 75
    cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 37
    cl::sycl::detail::pipe_reservation, 176
    cl::sycl::pipe_reservation, 188

k
    cl::sycl::detail::opencl_kernel, 553
kernel
    cl::sycl::detail::task, 591
    cl::sycl::kernel, 288, 289
kernel/detail/kernel.hpp
    TRISYCL_ParallelForKernel_RANGE, 1058
kernel_IR
    Device-side runtime implementation, 375
kernel_end
    cl::sycl::detail::queue, 567
kernel_exec
    opencl_spir.h, 742
kernel_start
    cl::sycl::detail::queue, 568
key_type
    cl::sycl::detail::cache, 506

LONG_MAX
    opencl_spir.h, 742
LONG_MIN
    opencl_spir.h, 743
latest_producer
    cl::sycl::detail::buffer_base, 121
latest_producer_mutex
    cl::sycl::detail::buffer_base, 122
linear_id
    Some helpers for the implementation, 361
lo
    opencl_spir.h, 775
local
    Dealing with OpenCL address spaces, 229
    Platforms, contexts, devices and queues, 331
local_index
    cl::sycl::nd_item, 442
local_mem_type
    Platforms, contexts, devices and queues, 330
local_ptr
    Dealing with OpenCL address spaces, 229
local_range
    cl::sycl::nd_range, 447

m
cl::sycl::detail::cache, 509
M_1_PI_F
opencl_spir.h, 743
M_1_PI
opencl_spir.h, 743
M_2_PI_F
opencl_spir.h, 743
M_2_PI
opencl_spir.h, 743
M_2_SQRTPI_F
opencl_spir.h, 744
M_2_SQRTPI
opencl_spir.h, 743
M_E_F
opencl_spir.h, 744
M_LN10
opencl_spir.h, 744
M_LN10_F
opencl_spir.h, 744
M_LN2
opencl_spir.h, 744
M_LN2_F
opencl_spir.h, 745
M_LOG10E_F
opencl_spir.h, 745
M_LOG10E
opencl_spir.h, 745
M_LOG2E_F
opencl_spir.h, 745
M_LOG2E
opencl_spir.h, 745
M_PI_2
opencl_spir.h, 746
M_PI_2_F
opencl_spir.h, 746
M_PI_4
opencl_spir.h, 746
M_PI_4_F
opencl_spir.h, 746
M_PI_F
opencl_spir.h, 746
M_PI
opencl_spir.h, 745
M_SQRT1_2
opencl_spir.h, 746
M_SQRT1_2_F
opencl_spir.h, 747
M_SQRT2
opencl_spir.h, 747
M_SQRT2_F
opencl_spir.h, 747
M_E
opencl_spir.h, 744
MAXFLOAT
opencl_spir.h, 747
make_id
Expressing parallelism through kernels, 452, 453
make_multi
Dealing with OpenCL address spaces, 231
make_range
Expressing parallelism through kernels, 453, 454
Manage default configuration and types, 367
__SYCL_SINGLE_SOURCE__, 367
CL_SYCL_LANGUAGE_VERSION, 367
SDX_KERNEL_PROGRAM_OWNING_BUG, 368
TRISYCL_CL_LANGUAGE_VERSION, 368
TRISYCL_DEVICE_ONLY, 368
TRISYCL_MAKE_BOOST_CIRCULARBUFFER↩
_THREAD_SAFE, 368
TRISYCL_SKIP_OPENCL, 368
map_allocator
Data access and storage in SYCL, 202
mark_as_written
cl::sycl::buffer, 140
cl::sycl::detail::buffer, 106
mask
opencl_spir.h, 775
math.hpp
TRISYCL_MATH_WRAP2, 1068
TRISYCL_MATH_WRAP2s, 1069
TRISYCL_MATH_WRAP3, 1069
TRISYCL_MATH_WRAP3s, 1069
TRISYCL_MATH_WRAP3ss, 1070
TRISYCL_MATH_WRAP, 1068
maxval
opencl_spir.h, 775
message
cl::sycl::exception, 382
min
cl::sycl, 486
minval
opencl_spir.h, 775
mode
cl::sycl::access, 489
cl::sycl::detail::pipe_accessor, 169
cl::sycl::detail::pipe_reservation, 184
modified
cl::sycl::detail::buffer, 111
move_read_reservation_forward
cl::sycl::detail::pipe, 150
move_write_reservation_forward
cl::sycl::detail::pipe, 151
multi_ptr
Dealing with OpenCL address spaces, 230
mutex_class
cl::sycl, 485

n
opencl_spir.h, 776
NAN
opencl_spir.h, 747
ND_range
cl::sycl::nd_item, 442
name
Platforms, contexts, devices and queues, 325
nd_item

cl::sycl::nd_item, 426, 427
nd_range
cl::sycl::nd_range, 444
ndr
cl::sycl::group, 415
non_const_value_type
cl::sycl::detail::buffer, 97
none
Platforms, contexts, devices and queues, 330, 331
notify_buffer_destructor
cl::sycl::detail::buffer_base, 122
notify_consumers
cl::sycl::detail::task, 582
num_elements
opencl_spir.h, 776
numa
Platforms, contexts, devices and queues, 327, 328
number_of_users
cl::sycl::detail::buffer_base, 122

offset
cl::sycl::item, 424
cl::sycl::nd_range, 447
opencl_spir.h, 776
ok
cl::sycl::detail::pipe_accessor, 170
cl::sycl::detail::pipe_reservation, 184
opencl
Platforms, contexts, devices and queues, 329
opencl_context
cl::sycl::detail::opencl_context, 535
opencl_device
cl::sycl::detail::opencl_device, 542
opencl_kernel
cl::sycl::detail::opencl_kernel, 550
opencl_kernel.hpp
TRISYCL_ParallelForKernel_RANGE, 1064
opencl_platform
cl::sycl::detail::opencl_platform, 296
opencl_queue
cl::sycl::detail::opencl_queue, 556
opencl_spir.h
__attribute__, 749, 761
__kernel_exec, 716
a, 772
as_char, 717
as_char16, 717
as_char2, 717
as_char3, 717
as_char4, 717
as_char8, 718
as_double, 718
as_double16, 718
as_double2, 718
as_double3, 718
as_double4, 718
as_double8, 719
as_float, 719
as_float16, 719
as_float2, 719
as_float3, 719
as_float4, 719
as_float8, 720
as_half, 720
as_half16, 720
as_half2, 720
as_half3, 720
as_half4, 720
as_half8, 721
as_int, 721
as_int16, 721
as_int2, 721
as_int3, 721
as_int4, 721
as_int8, 722
as_long, 722
as_long16, 722
as_long2, 722
as_long3, 722
as_long4, 722
as_long8, 723
as_short, 723
as_short16, 723
as_short2, 723
as_short3, 723
as_short4, 723
as_short8, 724
as_uchar, 724
as_uchar16, 724
as_uchar2, 724
as_uchar3, 724
as_uchar4, 724
as_uchar8, 725
as_uint, 725
as_uint16, 725
as_uint2, 725
as_uint3, 725
as_uint4, 725
as_uint8, 726
as_ulong, 726
as_ulong16, 726
as_ulong2, 726
as_ulong3, 726
as_ulong4, 726
as_ulong8, 727
as_ushort, 727
as_ushort16, 727
as_ushort2, 727
as_ushort3, 727
as_ushort4, 727
as_ushort8, 728
b, 773
c, 773
CHAR_BIT, 728
CHAR_MAX, 728
CHAR_MIN, 728
CLK_ADDRESS_CLAMP_TO_EDGE, 729

CLK_ADDRESS_CLAMP, 728
CLK_ADDRESS_MIRRORED_REPEAT, 729
CLK_ADDRESS_NONE, 729
CLK_ADDRESS_REPEAT, 729
CLK_ARGB, 729
CLK_BGRA, 730
CLK_DEPTH_STENCIL, 730
CLK_DEPTH, 730
CLK_FILTER_LINEAR, 730
CLK_FILTER_NEAREST, 730
CLK_FLOAT, 730
CLK_GLOBAL_MEM_FENCE, 731
CLK_HALF_FLOAT, 731
CLK_INTENSITY, 731
CLK_LOCAL_MEM_FENCE, 731
CLK_LUMINANCE, 731
CLK_NORMALIZED_COORDS_FALSE, 731
CLK_NORMALIZED_COORDS_TRUE, 732
CLK_RGBA, 733
CLK_RGBx, 733
CLK_RGB, 732
CLK_RGx, 733
CLK_RA, 732
CLK_RG, 732
CLK_Rx, 733
CLK_SIGNED_INT16, 733
CLK_SIGNED_INT32, 733
CLK_SIGNED_INT8, 734
CLK_SNORM_INT16, 734
CLK_SNORM_INT8, 734
CLK_UNORM_INT16, 734
CLK_UNORM_INT24, 734
CLK_UNORM_INT8, 734
CLK_UNORM_INT_101010, 735
CLK_UNORM_SHORT_555, 735
CLK_UNORM_SHORT_565, 735
CLK_UNSIGNED_INT16, 735
CLK_UNSIGNED_INT32, 735
CLK_UNSIGNED_INT8, 735
CLK_A, 728
CLK_R, 732
cl_mem_fence_flags, 760
cmp, 773
color, 773
const_func, 736
coord, 773
cosval, 773
DBL_DIG, 736
DBL_EPSILON, 736
DBL_MANT_DIG, 737
DBL_MAX_10_EXP, 737
DBL_MAX_EXP, 738
DBL_MAX, 737
DBL_MIN_10_EXP, 738
DBL_MIN_EXP, 739
DBL_MIN, 738
DBL_RADIX, 739
dst_stride, 774
edge1, 774
event, 774
event_list, 774
exp, 774
FLT_DIG, 739
FLT_EPSILON, 739
FLT_MANT_DIG, 739
FLT_MAX_10_EXP, 740
FLT_MAX_EXP, 740
FLT_MAX, 740
FLT_MIN_10_EXP, 740
FLT_MIN_EXP, 740
FLT_MIN, 740
FLT_RADIX, 741
FP_ILOGB0, 741
FP_ILOGBNAN, 741
HUGE_VALF, 741
HUGE_VAL, 741
i, 774
INFINITY, 742
INT_MAX, 742
INT_MIN, 742
intptr_t, 760
iptr, 775
kernel_exec, 742
LONG_MAX, 742
LONG_MIN, 743
lo, 775
M_1_PI_F, 743
M_1_PI, 743
M_2_PI_F, 743
M_2_PI, 743
M_2_SQRTPI_F, 744
M_2_SQRTPI, 743
M_E_F, 744
M_LN10, 744
M_LN10_F, 744
M_LN2, 744
M_LN2_F, 745
M_LOG10E_F, 745
M_LOG10E, 745
M_LOG2E_F, 745
M_LOG2E, 745
M_PI_2, 746
M_PI_2_F, 746
M_PI_4, 746
M_PI_4_F, 746
M_PI_F, 746
M_PI, 745
M_SQRT1_2, 746
M_SQRT1_2_F, 747
M_SQRT2, 747
M_SQRT2_F, 747
M_E, 744
MAXFLOAT, 747
mask, 775
maxval, 775
minval, 775

n, 776
NAN, 747
num_elements, 776
offset, 776
p, 776
p1, 776
printf, 772
quo, 777
readonly, 747
SCHAR_MAX, 748
SCHAR_MIN, 748
SHRT_MAX, 748
SHRT_MIN, 748
sampler, 777
signp, 777
src, 777
src_stride, 777
UCHAR_MAX, 748
UINT_MAX, 748
ULONG_MAX, 749
USHRT_MAX, 749
uchar, 760
uint, 761
uintptr_t, 761
ulong, 761
ushort, 761
val, 777
x, 778
y, 778
z, 778
opencl_type
cl::sycl::detail::address_space_base, 223
cl::sycl::detail::address_space_object, 217
cl::sycl::detail::address_space_variable, 225
opencl_types.hpp
TRISYCL_BOOST_COMPUTE_NAME, 1081
TRISYCL_DECLARE_CL_TYPES, 1081
TRISYCL_DEFINE_TYPES, 1081
TRISYCL_H_DEFINE_TYPE, 1082
TRISYCL_IS_WRAPPER_TRAIT, 1082
TRISYCL_SCALAR_TYPES, 1082
TRISYCL_SIZED_NAME, 1083
TRISYCL_TYPE_ACTUAL_NAME, 1083
TRISYCL_TYPE_CL_NAME, 1083
TRISYCL_TYPE_NAME, 1083
TRISYCL_TYPEDEF_TYPE, 1084
TRISYCL_WRAPPER_CLASS_2, 1084
TRISYCL_WRAPPER_CLASS_3, 1084
TRISYCL_WRAPPER_CLASS_4, 1084
operator BasicType
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 355
operator bool
cl::sycl::detail::pipe_accessor, 164
cl::sycl::detail::pipe_reservation, 182
cl::sycl::pipe_reservation, 194
operator FinalType
cl::sycl::detail::small_array, 349
operator opencl_type &
cl::sycl::detail::address_space_object, 218
cl::sycl::detail::address_space_variable, 226
operator<
cl::sycl::detail::shared_ptr_implementation, 573
operator<<
cl::sycl::detail::pipe_accessor, 165
operator>>
cl::sycl::detail::pipe_accessor, 165
operator*
cl::sycl::accessor, 59, 60
cl::sycl::detail::accessor, 86, 87
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 44, 45
operator()
cl::sycl::device_selector, 273
cl::sycl::device_type_selector, 268
std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >, 517
std::hash< cl::sycl::context >, 518
std::hash< cl::sycl::device >, 519
std::hash< cl::sycl::kernel >, 520
std::hash< cl::sycl::platform >, 521
std::hash< cl::sycl::queue >, 522
operator==
cl::sycl::detail::shared_ptr_implementation, 574
operator[]
cl::sycl::accessor, 60–64
cl::sycl::detail::accessor, 87–89
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 45–47
cl::sycl::detail::pipe_reservation, 182
cl::sycl::drt::accessor, 371
cl::sycl::group, 412
cl::sycl::item, 423
cl::sycl::pipe_reservation, 194
cl::sycl::xilinx::partition_array, 475
order
cl::sycl::detail::accessor_base, 33
owner_queue
cl::sycl::detail::task, 592

p
cl::sycl::detail::opencl_platform, 299
cl::sycl::detail::pipe_reservation, 184
cl::sycl::drt::code::program, 562
opencl_spir.h, 776
p1
opencl_spir.h, 776
parallel_OpenMP_for_iterate
cl::sycl::detail::parallel_OpenMP_for_iterate, 448
parallel_for
cl::sycl::handler, 277
Expressing parallelism through kernels, 454–457
parallel_for_global_offset
Expressing parallelism through kernels, 458
parallel_for_iterate
cl::sycl::detail::parallel_for_iterate, 448

cl::sycl::detail::parallel_for_iterate< 0, Range, ParallelForFunctor, Id >, 449
parallel_for_work_group
cl::sycl::handler, 277, 278
parallel_for_work_item
cl::sycl::group, 413, 414
Expressing parallelism through kernels, 459
parallel_for_workgroup
Expressing parallelism through kernels, 460
parallel_for_workitem
Expressing parallelism through kernels, 461
param_traits.hpp
TRISYCL_INFO_PARAM_TRAITS_ANY_T, 1043
TRISYCL_INFO_PARAM_TRAITS, 1043
partition_array
cl::sycl::xilinx::partition_array, 472, 473
partition_dim
cl::sycl::xilinx::partition::block, 504
cl::sycl::xilinx::partition::complete, 511
cl::sycl::xilinx::partition::cyclic, 513
partition_type
cl::sycl::xilinx::partition::block, 504
cl::sycl::xilinx::partition::complete, 511
cl::sycl::xilinx::partition::cyclic, 513
cl::sycl::xilinx::partition::none, 533
cl::sycl::xilinx::partition_array, 476
physical_mem_num
cl::sycl::xilinx::partition::cyclic, 513
pipe
cl::sycl::detail::pipe, 148
cl::sycl::pipe, 172
pipe_accessor
cl::sycl::detail::pipe_accessor, 162
pipe_reservation
cl::sycl::detail::pipe_reservation, 177, 178
cl::sycl::pipe_reservation, 189, 190
pipeline
cl::sycl::xilinx, 500
platform
cl::sycl::platform, 304, 305
Platforms, contexts, devices and queues, 324, 325, 331
platform_extensions
cl::sycl::detail::host_platform, 294
Platforms, contexts, devices and queues, 233
accelerator, 329
all, 329
automatic, 329
context, 323
cpu, 329
cpu_selector, 321
custom, 329
default_selector, 322
denorm, 329
device, 324
device::get_info< info::device::device_type >, 332
device::get_info< info::device::local_mem_size >, 332
device::get_info< info::device::max_compute_units >, 332
device::get_info< info::device::max_mem_alloc_size >, 333
device::get_info< info::device::max_work_group_size >, 333
device::get_info< info::device::name >, 333
device::get_info< info::device::profile >, 334
device::get_info< info::device::vendor >, 334
device_affinity_domain, 327
device_exec_capabilities, 322
device_execution_capabilities, 327
device_fp_config, 322
device_partition_property, 327
device_partition_type, 328
device_queue_properties, 323
device_type, 328
devices, 324
extensions, 325
fma, 330
fp_config, 329
get_devices, 334–336
global, 331
global_mem_cache_type, 330
gpu, 329
gpu_selector, 323
host, 329
host_selector, 323
local, 331
local_mem_type, 330
name, 325
none, 330, 331
numa, 327, 328
opencl, 329
platform, 324, 325, 331
profile, 325
TRISYCL_WEAK_ATTRIB_SUFFIX, 337
unsupported, 327, 328
vendor, 325
pointer
cl::sycl::detail::container_element_aspect, 344
cl::sycl::pipe_reservation, 188
pointer_t
cl::sycl::detail::address_space_ptr, 220
postlude
cl::sycl::detail::task, 583
prelude
cl::sycl::detail::task, 583
prevent_arguments_from_optimization
cl::sycl::detail, 495
printf
opencl_spir.h, 772
priv
Dealing with OpenCL address spaces, 230
private_ptr
Dealing with OpenCL address spaces, 230
producer_tasks
cl::sycl::detail::task, 592

profile
Platforms, contexts, devices and queues, 325
program
cl::sycl::drt::code::program, 561
prologues
cl::sycl::detail::task, 592

q
cl::sycl::detail::opencl_context, 540
cl::sycl::detail::opencl_queue, 560
queue
cl::sycl::detail::queue, 564
cl::sycl::info, 498
cl::sycl::queue, 312–316
queue_profiling
cl::sycl::info, 498
quo
opencl_spir.h, 777

r_rid_q
cl::sycl::detail::pipe, 158
rank
cl::sycl::detail::pipe_accessor, 170
rbegin
cl::sycl::accessor, 65
cl::sycl::detail::accessor, 90
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 48
cl::sycl::pipe_reservation, 195
read
cl::sycl::detail::pipe, 152
cl::sycl::detail::pipe_accessor, 166
read_done
cl::sycl::detail::pipe, 158
read_reserved_frozen
cl::sycl::detail::pipe, 159
readonly
opencl_spir.h, 747
ready
cl::sycl::detail::buffer_base, 122
cl::sycl::detail::reserve_id, 145
cl::sycl::detail::task, 592
ready_mutex
cl::sycl::detail::buffer_base, 122
cl::sycl::detail::task, 592
reference
cl::sycl::buffer, 128
cl::sycl::detail::accessor, 76
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 37
cl::sycl::detail::container_element_aspect, 344
cl::sycl::detail::pipe_accessor, 162
cl::sycl::detail::pipe_reservation, 176
cl::sycl::pipe_reservation, 188
reference_t
cl::sycl::detail::address_space_ptr, 221
register_accessor
cl::sycl::detail::accessor, 90
cl::sycl::detail::task, 584
release
cl::sycl::detail::buffer_base, 116
release_buffers
cl::sycl::detail::task, 585
remove
cl::sycl::detail::cache, 507
rend
cl::sycl::accessor, 65
cl::sycl::detail::accessor, 92
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 48
cl::sycl::pipe_reservation, 195
report_error
cl::sycl::error_handler, 378
cl::sycl::trisycl::default_error_handler, 515
reserve
cl::sycl::accessor< DataType, 1, AccessMode, access::target::blocking_pipe >, 71
cl::sycl::accessor< DataType, 1, AccessMode, access::target::pipe >, 68
cl::sycl::detail::pipe_accessor, 167
reserve_id
cl::sycl::detail::reserve_id, 144
reserve_read
cl::sycl::detail::pipe, 153
reserve_write
cl::sycl::detail::pipe, 154
reserved_for_reading
cl::sycl::detail::pipe, 155
reserved_for_writing
cl::sycl::detail::pipe, 155
reverse_iterator
cl::sycl::detail::accessor, 76
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 37
cl::sycl::pipe_reservation, 188
rid
cl::sycl::detail::pipe_reservation, 184
rid_iterator
cl::sycl::detail::pipe, 148
running_kernels
cl::sycl::detail::queue, 569

SCHAR_MAX
opencl_spir.h, 748
SCHAR_MIN
opencl_spir.h, 748
SDX_KERNEL_PROGRAM_OWNING_BUG
Manage default configuration and types, 368
SHRT_MAX
opencl_spir.h, 748
SHRT_MIN
opencl_spir.h, 748
sampler
opencl_spir.h, 777
schedule
cl::sycl::detail::task, 585
schedule_kernel
cl::sycl::handler, 279

select_device
cl::sycl::device_selector, 273
serialize_accessor_arg
Device-side runtime implementation, 372
serialize_arg
Device-side runtime implementation, 373
set
cl::sycl::item, 423
set_arg
cl::sycl::detail::task, 587, 588
cl::sycl::handler, 279–281
set_args
cl::sycl::handler, 281
set_debug
cl::sycl::detail::pipe_accessor, 167
set_final_data
cl::sycl::buffer, 140–142
cl::sycl::detail::buffer, 107, 108
set_global
cl::sycl::nd_item, 441
set_kernel
cl::sycl::detail::task, 589
Device-side runtime implementation, 374
set_kernel_task_marker
cl::sycl::detail, 494
set_latest_producer
cl::sycl::detail::buffer_base, 117
set_local
cl::sycl::nd_item, 441
set_order
cl::sycl::detail::accessor_base, 32
shared_ptr_class
cl::sycl, 485
shared_ptr_implementation
cl::sycl::detail::shared_ptr_implementation, 572
cl::sycl::platform, 309
signp
opencl_spir.h, 777
single_task
cl::sycl::detail::kernel, 286
cl::sycl::detail::opencl_kernel, 552
cl::sycl::handler, 282
size
cl::sycl::detail::pipe, 156
cl::sycl::detail::pipe_accessor, 168
cl::sycl::detail::pipe_reservation, 183
cl::sycl::detail::reserve_id, 145
cl::sycl::drt::accessor, 372
cl::sycl::pipe_reservation, 196
cl::sycl::xilinx::partition_array, 475
size_type
cl::sycl::pipe_reservation, 189
size_with_lock
cl::sycl::detail::pipe, 156
small_array
cl::sycl::detail::small_array, 347, 348
small_array_123
cl::sycl::detail::small_array_123< BasicType, FinalType, 1 >, 354, 355
cl::sycl::detail::small_array_123< BasicType, FinalType, 2 >, 356, 357
cl::sycl::detail::small_array_123< BasicType, FinalType, 3 >, 359, 360
Some helpers for the implementation, 343
linear_id, 361
TRISYCL_BOOST_OPERATOR_VECTOR_OP, 360
TRISYCL_LOGICAL_OPERATOR_VECTOR_OP, 360
unimplemented, 362
src
opencl_spir.h, 777
src_stride
opencl_spir.h, 777
start
cl::sycl::detail::reserve_id, 145
static_pipe
cl::sycl::static_pipe, 199
std, 502
std::hash< cl::sycl::buffer< T, Dimensions, Allocator > >, 516
operator(), 517
std::hash< cl::sycl::context >, 517
operator(), 518
std::hash< cl::sycl::device >, 518
operator(), 519
std::hash< cl::sycl::kernel >, 519
operator(), 520
std::hash< cl::sycl::platform >, 520
operator(), 521
std::hash< cl::sycl::queue >, 521
operator(), 522
string_class
cl::sycl, 485
submit
cl::sycl::queue, 318, 319
super
cl::sycl::detail::address_space_array, 211
cl::sycl::detail::address_space_fundamental, 214
cl::sycl::detail::address_space_ptr, 221
cl::sycl::detail::address_space_variable, 225
sync_with_host
cl::sycl::detail::buffer_base, 117

TRISYCL_BOOST_COMPUTE_NAME
opencl_types.hpp, 1081
TRISYCL_BOOST_OPERATOR_VECTOR_OP
Some helpers for the implementation, 360
TRISYCL_CL_LANGUAGE_VERSION
Manage default configuration and types, 368
TRISYCL_CONSTANT_AS
address_space/detail/address_space.hpp, 621
TRISYCL_DECLARE_CL_TYPES
opencl_types.hpp, 1081
TRISYCL_DEFINE_TYPES
opencl_types.hpp, 1081

TRISYCL_DEFINE_VEC_TYPE_SIZE
Vector types in SYCL, 469
TRISYCL_DEFINE_VEC_TYPE
Vector types in SYCL, 468
TRISYCL_DEVICE_ONLY
Manage default configuration and types, 368
TRISYCL_DUMP_T
debug.hpp, 687
TRISYCL_DUMP
debug.hpp, 686
TRISYCL_GENERIC_AS
address_space/detail/address_space.hpp, 622
TRISYCL_GLOBAL_AS
address_space/detail/address_space.hpp, 622
TRISYCL_H_DEFINE_TYPE
opencl_types.hpp, 1082
TRISYCL_INFO_PARAM_TRAITS_ANY_T
param_traits.hpp, 1043
TRISYCL_INFO_PARAM_TRAITS
param_traits.hpp, 1043
TRISYCL_INTERNAL_DUMP
debug.hpp, 687
TRISYCL_IS_WRAPPER_TRAIT
opencl_types.hpp, 1082
TRISYCL_LOCAL_AS
address_space/detail/address_space.hpp, 622
TRISYCL_LOGICAL_OPERATOR_VECTOR_OP
Some helpers for the implementation, 360
TRISYCL_MAKE_BOOST_CIRCULARBUFFER_TH↩READ_SAFE
Manage default configuration and types, 368
TRISYCL_MATH_WRAP2
math.hpp, 1068
TRISYCL_MATH_WRAP2s
cl::sycl, 486
math.hpp, 1069
TRISYCL_MATH_WRAP3
math.hpp, 1069
TRISYCL_MATH_WRAP3s
math.hpp, 1069
TRISYCL_MATH_WRAP3ss
math.hpp, 1070
TRISYCL_MATH_WRAP
cl::sycl, 486
math.hpp, 1068
TRISYCL_PRIVATE_AS
address_space/detail/address_space.hpp, 622
TRISYCL_ParallelForFunctor_GLOBAL_OFFSET
handler.hpp, 1029
TRISYCL_ParallelForKernel_RANGE
cl::sycl::detail::kernel, 286
cl::sycl::detail::opencl_kernel, 552
handler.hpp, 1030
kernel/detail/kernel.hpp, 1058
opencl_kernel.hpp, 1064
TRISYCL_SCALAR_TYPES
opencl_types.hpp, 1082
TRISYCL_SIZED_NAME
opencl_types.hpp, 1083
TRISYCL_SKIP_OPENCL
Manage default configuration and types, 368
TRISYCL_TYPE_ACTUAL_NAME
opencl_types.hpp, 1083
TRISYCL_TYPE_CL_NAME
opencl_types.hpp, 1083
TRISYCL_TYPE_NAME
opencl_types.hpp, 1083
TRISYCL_TYPEDEF_TYPE
opencl_types.hpp, 1084
TRISYCL_WEAK_ATTRIB_PREFIX
global_config.hpp, 695
TRISYCL_WEAK_ATTRIB_SUFFIX
global_config.hpp, 695
Platforms, contexts, devices and queues, 337
TRISYCL_WRAPPER_CLASS_2
opencl_types.hpp, 1084
TRISYCL_WRAPPER_CLASS_3
opencl_types.hpp, 1084
TRISYCL_WRAPPER_CLASS_4
opencl_types.hpp, 1084
TRISYCL_parallel_for_functor_GLOBAL
cl::sycl::handler, 283
handler.hpp, 1029
target
cl::sycl::access, 489
cl::sycl::detail::pipe_accessor, 170
cl::sycl::detail::pipe_reservation, 185
task
cl::sycl::detail::accessor_base, 33
cl::sycl::detail::task, 579
cl::sycl::handler, 284
throw_asynchronous
cl::sycl::queue, 319
trace_kernel
Debugging and tracing support, 365
track_access_mode
cl::sycl::detail::buffer, 108
tuple_to_array
Helpers to do array and tuple conversion, 341
tuple_to_array_iterate
Helpers to do array and tuple conversion, 341
type
cl::sycl::detail::address_space_base, 223
cl::sycl::detail::ocl_type, 207
cl::sycl::detail::ocl_type< T, constant_address_↩space >, 207
cl::sycl::detail::ocl_type< T, global_address_space >, 208
cl::sycl::detail::ocl_type< T, local_address_space >, 208
cl::sycl::detail::ocl_type< T, private_address_↩space >, 209
cl::sycl::device, 265
cl::sycl::xilinx::partition, 502

UCHAR_MAX
opencl_spir.h, 748

UINT_MAX
opencl_spir.h, 748
ULONG_MAX
opencl_spir.h, 749
USHRT_MAX
opencl_spir.h, 749
uchar
opencl_spir.h, 760
uint
opencl_spir.h, 761
uintptr_t
opencl_spir.h, 761
ulong
opencl_spir.h, 761
unimplemented
Some helpers for the implementation, 362
unique_ptr_class
cl::sycl, 485
unsupported
Platforms, contexts, devices and queues, 327, 328
update_buffer_state
cl::sycl::detail::buffer_base, 118
use
cl::sycl::detail::buffer_base, 120
use_count
cl::sycl::buffer, 142
used_for_reading
cl::sycl::detail::pipe, 159
used_for_writing
cl::sycl::detail::pipe, 159
ushort
opencl_spir.h, 761

val
opencl_spir.h, 777
value_type
cl::sycl::buffer, 129
cl::sycl::detail::accessor, 76
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
cl::sycl::detail::buffer, 97
cl::sycl::detail::cache, 506
cl::sycl::detail::container_element_aspect, 344
cl::sycl::detail::pipe, 148
cl::sycl::detail::pipe_accessor, 162
cl::sycl::detail::pipe_reservation, 176
cl::sycl::pipe, 172
cl::sycl::pipe_reservation, 189
cl::sycl::static_pipe, 199
variable
cl::sycl::detail::address_space_variable, 227
vec
cl::sycl::vec, 466, 467
Vector types in SYCL, 464
TRISYCL_DEFINE_VEC_TYPE_SIZE, 469
TRISYCL_DEFINE_VEC_TYPE, 468
vector_class
cl::sycl, 485
vendor
Platforms, contexts, devices and queues, 325

w_rid_q
cl::sycl::detail::pipe, 159
wait
cl::sycl::detail::buffer_base, 120
cl::sycl::detail::task, 589
cl::sycl::queue, 320
wait_and_throw
cl::sycl::queue, 320
wait_for_kernel_execution
cl::sycl::detail::queue, 568
wait_for_producers
cl::sycl::detail::task, 590
waiter
Data access and storage in SYCL, 204
weak_ptr_class
cl::sycl, 486
what
cl::sycl::exception, 381
writable_array_type
cl::sycl::detail::accessor< T, Dimensions, Mode, access::target::local >, 38
writable_array_view_type
cl::sycl::detail::accessor, 76
write
cl::sycl::detail::pipe, 157
cl::sycl::detail::pipe_accessor, 168
write_done
cl::sycl::detail::pipe, 159

x
cl::sycl::detail::small_array, 349
opencl_spir.h, 778
Xilinx vendor extensions, 470
_ssdm_SpecArrayPartition, 477
_ssdm_op_SpecDataflowPipeline, 477
_ssdm_op_SpecPipeline, 477

y
cl::sycl, 487
cl::sycl::detail::small_array, 350
opencl_spir.h, 778

z
cl::sycl, 487
cl::sycl::detail::small_array, 351
opencl_spir.h, 778