APPENDIX A REFERENCE GUIDE

A.1 MAP Resources

```
int map_allocate (int nmaps);

int map_free (int nmaps);
```

A.2 Declaring Arrays Used in DMA Operations

```
void *Cache_Aligned_Allocate(int size);

void Cache_Aligned_Free(char *buffer);
```

A.3 Barriers

A.3.1 MAP Processors and CPU Synchronization

```
Barrier_group_t Barrier_Allocate (void);

void Barrier_Initialize (Barrier_group_t barr, int id, int num);

void Barrier_Wait (Barrier_group_t barr, int id);

void Timed_Barrier_Wait (Barrier_group_t barr, int id, int time_ms);
```

A.3.2 In-Chip Barriers

```
In_Chip_Barrier <barrier_name>;

void In_Chip_Barrier_Set (In_Chip_Barrier*, int);

void In_Chip_Barrier_Wait (In_Chip_Barrier*);
```

A.3.3 In-Chip Locks

```
In_Chip_Lock <lock_name>;

void In_Chip_Lock_Acquire (In_Chip_Lock*, int);

void In_Chip_Lock_Release (In_Chip_Lock*);
```

A.4 Pure Functional Macros

```
void imax (int in1, int in2, int *result);

void imin (int in1, int in2, int *result);

void isqrt_32 (int in1, int *result);

void median_8_9 (int8_t in1, int8_t in2, int8_t in3, int8_t in4, int8_t in5, int8_t in6,
                 int8_t in7, int8_t in8, int8_t in9, int8_t *result);

void popcount_32 (int32_t input, int *result);
```

```
void popcount_64 (int64_t input, int *result);

void selector_8  (int selector, int8_t v1, int8_t v2, int8_t  *result);

void selector_16 (int selector, int16_t v1, int16_t v2, int16_t *result);

void selector_32 (int selector, int32_t v1, int32_t v2, int32_t *result);

void selector_64 (int selector, int64_t v1, int64_t v2, int64_t *result);

void bitrev_32 (int val, int nbits, int *result);
```

A.5 Extended Precision Integer Add

The *add_extended* function is a 64-bit integer add with carry bits in and out. This allows the user to create arbitrarily wide integer adds by chaining the *carry out* of one call to the *carry in* of the next call. For example, to do a 128-bit integer add:

```
uint64_t ahi, alo, bhi, blo, rhi, rlo;
uint32_t carry0, carry1;
...
add_extended (alo, blo, 0, &rlo, &carry0);
add_extended (ahi, bhi, carry0, &rhi, &carry1);
...
```

The first operand lives in *alo* and *ahi*, the second operand in *blo* and *bhi*, and the result goes into *rlo* and *rhi*. These are fully pipelined. If a chain of *add_extended* calls exists in a pipelined loop, a very wide add may be done in every clock tick.

The prototype for this function is:

```
void add_extended (uint64_t v0, uint64_t v1, uint32_t cin, uint64_t *vout, uint32_t *cout);
```

A.6 Display Macros

The following display macros provide a way to generate data values with a constant marker at every simulated clock tick, depending on the whether the value of the last argument is evaluated to 1 (true) or 0 (false):

```
vdisplay_float (float val, int marker, int valid);
vdisplay_double (double val, int marker, int valid);
vdisplay_8 (int8_t val, int marker, int valid);
vdisplay_16 (int16_t val , int marker, int valid);
vdisplay_32 (int32_t val, int marker, int valid);
vdisplay_64 (int64_t val, int marker, int valid);
```

A.7 Single Precision Macros

```
float fabsf  (float a);

float sqrtf  (float a);

float fminf  (float v1, float v2);

float fmaxf  (float v1, float v2);
```

```
float sinf   (float a);

float cosf   (float a);

float tanf   (float a);

float atanf  (float a);

float atan2f (float v1, float v2);

float sinhf  (float a);

float coshf  (float a);

float tanhf  (float a);

float expf   (float a);

float powf   (float a, float b);

float logf   (float a);
```

A.8 Double Precision Macros

```
double fabs (double a);

double sqrt (double a);

double fmin (double v1, double v2);

double fmax (double v1, double v2);
```

A.9 Accumulators

Accumulators are a family of stateful macros used in pipelined loops. They are often used to avoid loop slowdowns caused by loop-carried scalar dependencies and come in a variety of reduction functions: *add*, *bit-and*, *bit-or*, *bit-eor*, *min*, *max*, *umin*, and *umax*.

Accumulators are flexible in their use as each has an explicit reset input to determine when its internal state is cleared and an explicit reset value that specifies the starting state of the accumulator. There is an enable input that determines whether the current input value is accumulated. In the case where both the reset and the enable inputs are true, the accumulator's state is set to the reset value and immediately updated with the current input value.

Each accumulator has two versions: <*name*> and <*name*>*_np*. A macro of the first form stops its internal accumulation when loop termination is detected so that the accumulation may be continued if the loop is reentered later. This behavior is appropriate for most situations. If the loop termination test is dependent on the output value of the accumulator, loop slowdown occurs because a new iteration cannot be launched until the current accumulator value is available. In that case, the <*name*>*_np* version may be used. That version allows the accumulator to free run beyond loop termination, breaking the cyclic dependency and avoiding loop slowdown. Note that the state of the accumulator is some unpredictable value if the loop is later reentered. When using the <*name*>*_np* form of an accumulator, one always resets the accumulator at the start of the loop.

A.9.1 Integer Accumulators

```
cg_accum_<xxx>_32 (int32_t val, int enable, int rset_val, int rset_enable, int *result);

cg_accum_<xxx>_64 (int64_t val, int enable, int64_t rset_val, int rset_enable, int64_t
                              *result);
```

Where the <*xxx*> is one of the values in Table 26.

Table 26 Integer Accumulator Operations

Operation	Description
add	Integer addition
bit_and	Bitwise ANDing
bit_or	Bitwise ORing
bit_eor	Bitwise exclusive ORing
imax	Signed int maximum
imin	Signed int minimum
umax	Unsigned int maximum
umin	Unsigned int minimum

A.9.2 Floating Point Accumulators

A.9.2.1 Floating Point Sum Accumulator

This simplest floating point accumulator conditionally sums a floating point value into the result, in the same manner as the integer *cg_acccum_add* macros. An accumulated sum is output only when the value of the argument *last* is non-zero. The prototypes for the floating point add accumulate macros are:

```
void fp_accum_32 (float val, int last, int enable, int reset, float *result, int *err);

void fp_accum_64 (double val, int last, int enable, int reset, double *result, int *err);
```

A.9.3 Floating Point Multiply Sum Accumulator

The floating point multiply sum accumulator multiplies two input values together and applies the product to the sum accumulator based on the value of *enable*. If enable is non-zero, the product is applied to the sum. If the enable is zero, the product is not applied to the sum. The result value is output only when the value of *last* is true non-zero. The prototypes for the multiply sum accumulators are:

```
void fp_mac_32 (float v1, float v2, int last, int enable, int reset, float *result, int *err);

void fp_mac_64 (double v1, double v1, int last, int enable, int reset, double *result, int
                              *err);
```

A.10 Counters

Counters are related to accumulators. They increment by one but have a specified ceiling value. Once the ceiling is reached, the macro falls back to its reset value. Note that these functions can be at the top of the loop because, unlike the accumulators, the reset enable overrides the other enable, so in the first iteration, the counters zero but do not increment.

```
cg_count_ceil_32 (int enable, int rset_val, int rset_enable, int ceiling_val, int *result);
```

```
cg_count_ceil_64 (int enable, int64_t rset_val, int rset_enable, int ceiling_val, int64_t
                              *result);
```

A.11 Sequence Generators

The sequence generator *cg_incr_seq_32* produces sequences of *ints*, where the max of each sequence is one greater than the max of the previous sequence:

```
cg_incr_seq_32 (int a, int b, int *result)
```

The sequence generator *cg_decr_seq_32* produces a sequence of *ints*, where the number of values in each subsequence is one less than in the previous subsequence.

```
cg_decr_seq_32 (int enable, int reset, int length, int *result);
```

A.12 Loop-Carried Macros (Delay Queues)

```
delay_queue_8_16   (int8_t  val_in, int enable, int reset, int8_t  *val_out);

delay_queue_16_16  (int16_t val_in, int enable, int reset, int16_t *val_out);

delay_queue_32_16  (int32_t val_in, int enable, int reset, int32_t *val_out);

delay_queue_64_16  (int64_t val_in, int enable, int reset, int64_t *val_out);

delay_queue_8_var  (int8_t val_in, int enable, int reset, int queue_len, int8_t *val_out);

delay_queue_16_var (int16_t val_in, int enable, int reset, int queue_len, int16_t *val);

delay_queue_32_var (int32_t val_in, int enable, int reset, int queue_len, int32_t *val);

delay_queue_64_var (int64_t val_in, int enable, int reset, int queue_len, int64_t *val);
```

A.13 Timing Macros

These timing macros start and then check the status of a counter accumulating the number of clocks during execution on a MAP processor:

```
start_timer ();

read_timer (int64_t *r);
```

This timing macro provides delays execution on a MAP processor by *v* number of clock ticks before continuing on:

```
spin_wait (int v);
```

A.14 Split and Combine Macros

A.14.1 Split Macros

```
split_64to4  (int64_t a, int8_t *r15, int8_t *r14, int8_t *r13, int8_t *r12, int8_t *r11,
                         int8_t *r10, int8_t *r9, int8_t *r8, int8_t *r7, int8_t *r6,
                         int8_t *r5, int8_t *r4, int8_t *r3, int8_t *r2, int8_t *r1,
                         int8_t *r0);
```

```
split_64to8  (int64_t a, int8_t *r7, int8_t *r6, int8_t *r5, int8_t *r4, int8_t *r3, int8_t
                          *r2, int8_t *r1, int8_t *r0);

split_64to16 (int64_t a, int16_t *r3, int16_t *r2, int16_t *r1, int16_t *r0);

split_64to32 (int64_t a, int *r1, int *r0);

split_32to4  (int a, int8_t *r7, int8_t *r6, int8_t *r5, int8_t *r4, int8_t *r3, int8_t *r2,
                          int8_t *r1, int8_t *r0);

split_32to8  (int a, int8_t *r3, int8_t *r2, int8_t *r1, int8_t *r0);

split_32to16 (int a, int16_t *r1, int16_t *r0);

split_16to4  (int16_t a, int8_t *r3, int8_t *r2, int8_t *r1, int8_t *r0);

split_16to8  (int16_t a, int8_t *r1, int8_t *r0);

split_8to4   (int8_t a, int8_t *r1, int8_t *r0);
```

A.14.2 Combine Macros

```
comb_4to64 (int8_t a15, int8_t a14, int8_t a13, int8_t a12, int8_t a11, int8_t a10, int8_t a9,
                          int8_t a8, int8_t a7, int8_t a6, int8_t a5, int8_t a4, int8_t a3,
                          int8_t a2, int8_t a1, int8_t a0, int64_t *r);

comb_8to64 (int8_t a7, int8_t a6, int8_t a5, int8_t a4, int8_t a3, int8_t a2, int8_t a1,
                          int8_t a0, int64_t *r);

comb_16to64 (int16_t a3, int16_t a2, int16_t a1, int16_t a0, int64_t *r);

comb_32to64 (int a1, int a0, int64_t *r);

comb_4to32  (int8_t a7, int8_t a6, int8_t a5, int8_t a4, int8_t a3, int8_t a2, int8_t a1,
                          int8_t a0, int *r);

comb_8to32  (int8_t a3, int8_t a2, int8_t a1, int8_t a0, int *r);

comb_16to32 (int16_t a1, int16_t a0, int *r);

comb_4to16  (int8_t a3, int8_t a2, int8_t a1, int8_t a0, int16_t *r);

comb_8to16  (int8_t a1, int8_t a0, int16_t *r);

comb_4to8   (int8_t a1, int8_t a0, int8_t *r);
```

A.14.3 Floating Point Split and Combine Macros

Splitting and combining using float values requires special functions to ensure that unintended numeric conversion does not occur. The following functions deal with a pair of floats packed in a word, as well as a combined int and float:

```
split_64to32_flt_flt (int64_t a, float *r1, float *r0);

split_64to32_flt_int (int64_t a, float *r1, int *r0);
```

```
split_64to32_int_flt (int64_t a, int *r1, float *r0);

comb_32to64_flt_flt  (float a1, float a0, int64_t *r);

comb_32to64_flt_int  (float a1, int a0, int64_t *r);

comb_32to64_int_flt  (int a1, float a0, int64_t *r);
```

A.15 Stream Functions

A.15.1 Producer Functions

The following functions send values to a stream dependent on an enable bit:

```
void put_stream_8        (Stream_8* S, int8_t val, int enable);

void put_stream_16       (Stream_16* S, int16_t val, int enable);

void put_stream_32       (Stream_32* S, int32_t val, int enable);

void put_stream_64       (Stream_64* S, int64_t val, int enable);

void put_stream_128      (Stream_128* S, int64_t val1, int64_t val2, int enable);

void put_stream_192      (Stream_192* S, int64_t val1, int64_t val2, int64_t val3, int
                             enable);

void put_stream_256      (Stream_256* S, int64_t val1, int64_t val2, int64_t val3, int64_t
                             val4, int enable);

void put_stream_320      (Stream_320* S, int64_t val1, int64_t val2, int64_t val3, int64_t
                             val4, int64_t val5, int enable);

void put_stream_512      (Stream_512* S, int64_t val1, int64_t val2, int64_t val3, int64_t
                             val4, int64_t val5, int64_t val6, int64_t val7, int64_t val8, int
                             enable);

void put_stream_1024     (Stream_1024* S, int64_t val1, int64_t val2, int64_t val3, int64_t
                             val4, int64_t val5, int64_t val6, int64_t val7, int64_t val8,
                             int64_t val9, int64_t val10, int64_t val11, int64_t val12,
                             int64_t val13, int64_t val14, int64_t val15, int64_t val16, int
                             enable);

void put_stream_flt_32   (Stream_32* S, float val, int enable);

void put_stream_dbl_64   (Stream_64* S, double val, int enable);

void put_stream_dbl_128  (Stream_128* S, double val1, double val2, int enable);

void put_stream_dbl_192  (Stream_192* S, double val1, double val2, double val3, int enable);

void put_stream_dbl_256  (Stream_256* S, double val1, double val2, double val3, double val4,
                             int enable);

void put_stream_dbl_320  (Stream_320* S, double val1, double val2, double val3, double val4,
                             double val5, int enable);
```

```
void put_stream_dbl_512  (Stream_512* S, double val1, double val2, double val3, double val4,
                              double val5, double val6, double val7, double val8, int enable);

void put_stream_dbl_1024 (Stream_1024* S, double val1, double val2, double val3, double val4,
                              double val5, double val6, double val7, double val8, double val9,
                              double val10, double val11, double val12, double val13, double
                              val14, double val15, double val16, int enable);
```

A.15.2 Consumer Functions

These functions retrieve values from a stream:

```
void get_stream_8        (Stream_8* S, int8_t* val);

void get_stream_16       (Stream_16* S, int16_t* val);

void get_stream_32       (Stream_32* S, int32_t* val);

void get_stream_64       (Stream_64* S, int64_t* val);

void get_stream_128      (Stream_128* S, void* val1, void* val2);

void get_stream_192      (Stream_192* S, void* val1, void* val2, void* val3);

void get_stream_256      (Stream_256* S, void* val1, void* val2, void* val3, void* val4);

void get_stream_320      (Stream_320* S, void* val1, void* val2, void* val3, void* val4, void*
                              val5);

void get_stream_512      (Stream_512* S, void* val1, void* val2, void* val3, void* val4, void*
                              val5, void* val6, void* val7, void* val8);

void get_stream_1024     (Stream_1024* S, void* val1, void*val2, void* val3, void* val4, void*
                              val5, void* val6, void* val7, void* val8, void* val9, void*
                              val10, void* val11, void* val12, void* val13, void*val14, void*
                              val15, void* val16);

void get_stream_flt_32   (Stream_32* S, float* val);

void get_stream_dbl_64   (Stream_64* S, double* val);

void get_stream_dbl_128  (Stream_128* S, double* val1, double* val2);

void get_stream_dbl_192  (Stream_192* S, double* val1, double* val2, double* val3);

void get_stream_dbl_256  (Stream_256* S, double* val1, double* val2, double* val3, double*
                              val4);

void get_stream_dbl_320  (Stream_320* S, double* val1, double* val2, double* val3, double*
                              val4, double* val5);

void get_stream_dbl_512  (Stream_512* S, double* val1, double* val2, double* val3, double*
                              val4, double* val5, double* val6, double* val7, double* val8);
```

```
void get_stream_dbl_1024 (Stream_1024512** S, double* val1, double* val2, double* val3,
                          double* val4, double* val5, double* val6, double* val7, double*
                          val8, double* val9, double* val10, double* val11, double* val12,
                          double* val13, double* val14, double* val15, double* val16);
```

A.15.3 Stream Termination

```
void stream_8_term    (Stream_8 *S);

void stream_16_term   (Stream_16 *S);

void stream_32_term   (Stream_32 *S);

void stream_64_term   (Stream_64 *S);

void stream_128_term  (Stream_128 *S);

void stream_192_term  (Stream_192 *S);

void stream_256_term  (Stream_256 *S);

void stream_320_term  (Stream_320 *S);

void stream_512_term  (Stream_512 *S);

void stream_1024_term (Stream_1024 *S);
```

A.15.4 Testing for Stream Termination

```
int all_streams_active    (void);

int is_stream_8_active    (Stream_8 *S);

int is_stream_16_active   (Stream_16 *S);

int is_stream_32_active   (Stream_32 *S);

int is_stream_64_active   (Stream_64 *S);

int is_stream_128_active  (Stream_128 *S);

int is_stream_192_active  (Stream_192 *S);

int is_stream_256_active  (Stream_256 *S);

int is_stream_320_active  (Stream_320 *S);

int is_stream_512_active  (Stream_512 *S);

int is_stream_1024_active (Stream_1024 *S);
```

A.15.5 Stream Histogram

```
void stream_hist_8_term         (Stream_8 *Source, Stream_32 *Results, int frame_size);

void stream_hist_full_8_term    (Stream_8 *Source, Stream_32 *Results, int frame_size);

void stream_hist_full_8_64_term (Stream_64 *Source, Stream_32 *Results, int frame_size);
```

A.15.6 Stream Merges

A.15.6.1 Round Robin Merges

```
void stream_merge_2_8       (Stream_8 *S1, Stream_8 *S2, Stream_8 *SM, int64_t size);

void stream_merge_2_16      (Stream_16 *S1, Stream_16 *S2, Stream_16 *SM, int64_t size);

void stream_merge_2_32      (Stream_32 *S1, Stream_32 *S2, Stream_32 *SM, int64_t size);

void stream_merge_2_64      (Stream_64 *S1, Stream_64 *S2, Stream_64 *SM, int64_t size);

void stream_merge_3_8       (Stream_8 *S1, Stream_8 *S2, Stream_8 *S3, Stream_8 *SM, int64_t
                             size);

void stream_merge_3_16      (Stream_16 *S1, Stream_16 *S2, Stream_16 *S3, Stream_16 *SM,
                             int64_t size);

void stream_merge_3_32      (Stream_32 *S1, Stream_32 *S2, Stream_32 *S3, Stream_32 *SM,
                             int64_t size);

void stream_merge_3_64      (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, Stream_64 *SM,
                             int64_t size);

void stream_merge_4_8       (Stream_8 *S1, Stream_8 *S2, Stream_8 *S3, Stream_8 *S4, Stream_8
                             *SM, int64_t size);

void stream_merge_4_16      (Stream_16 *S1, Stream_16 *S2, Stream_16 *S3, Stream_16 *S4,
                             Stream_16 *SM, int64_t size);

void stream_merge_4_32      (Stream_32 *S1, Stream_32 *S2, Stream_32 *S3, Stream_32 *S4,
                             Stream_32 *SM, int64_t size);

void stream_merge_4_64      (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, Stream_64 *S4,
                             Stream_64 *SM, int64_t size);

void stream_merge_5_8       (Stream_8 *S1, Stream_8 *S2, Stream_8 *S3, Stream_8 *S4, Stream_8
                             *S4, Stream_8 *S5, int64_t size);

void stream_merge_5_16      (Stream_16 *S1, Stream_16 *S2, Stream_16 *S3, Stream_16 *S4,
                             Stream_16 *S5, Stream_16 *SM, int64_t size);

void stream_merge_5_32      (Stream_32 *S1, Stream_32 *S2, Stream_32 *S3, Stream_32 *S4,
                             Stream_32 *S5, Stream_32 *SM, int64_t size);

void stream_merge_5_64      (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, Stream_64 *S4,
                             Stream_64 *S5, Stream_64 *SM, int64_t size);
```

```
void stream_merge_2_8_term  (Stream_8 *S1, Stream_8 *S2, Stream_8 *SM);

void stream_merge_2_16_term (Stream_16 *S1, Stream_16 *S2, Stream_16 *SM);

void stream_merge_2_32_term (Stream_32 *S1, Stream_32 *S2, Stream_32 *SM);

void stream_merge_2_64_term (Stream_64 *S1, Stream_64 *S2, Stream_64 *SM);

void stream_merge_3_8_term  (Stream_8 *S1, Stream_8 *S2, Stream_8 *S3, Stream_8 *SM);

void stream_merge_3_16_term (Stream_16 *S1, Stream_16 *S2,Stream_16 *S3, Stream_16 *SM);

void stream_merge_3_32_term (Stream_32 *S1, Stream_32 *S2,Stream_32 *S3, Stream_32 *SM);

void stream_merge_3_64_term (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, Stream_64 *SM);

void stream_merge_4_8_term  (Stream_8 *S1, Stream_8 *S2, Stream_8 *S3, Stream_8 *S4, Stream_8
                             *SM);

void stream_merge_4_16_term (Stream_16 *S1, Stream_16 *S2, Stream_16 *S3, Stream_16 *S4,
                             Stream_16 *SM);

void stream_merge_4_32_term (Stream_32 *S1, Stream_32 *S2, Stream_32 *S3, Stream_32 *S4,
                             Stream_32 *SM);

void stream_merge_4_64_term (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, Stream_64 *S4,
                             Stream_64 *SM);

void stream_merge_5_8_term  (Stream_8 *S1, Stream_8 *S2, Stream_8 *S3, Stream_8 *S4, Stream_8
                             *S5, Stream_8 *SM);

void stream_merge_5_16_term (Stream_16 *S1, Stream_16 *S2, Stream_16 *S3, Stream_16 *S4,
                             Stream_16 *S5, Stream_16 *SM);

void stream_merge_5_32_term (Stream_32 *S1, Stream_32 *S2,Stream_32 *S3, Stream_32 *S4,
                             Stream_32 *S5, Stream_32 *SM);

void stream_merge_5_64_term (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, Stream_64 *S4,
                             Stream_64 *S5, Stream_64 *SM);
```

A.15.6.2 Non-Deterministic Merges

```
void stream_merge_nd_2_8_term    (Stream_8 *S1, Stream_8 *S2, Stream_8 *SM);

void stream_merge_nd_2_16_term   (Stream_16 *S1, Stream_16 *S2, Stream_16 *SM);

void stream_merge_nd_2_32_term   (Stream_32 *S1, Stream_32 *S2, Stream_32 *SM);

void stream_merge_nd_2_64_term   (Stream_64 *S1, Stream_64 *S2, Stream_64 *SM);

void stream_merge_nd_2_128_term  (Stream_128 *S1, Stream_128 *S2, Stream_128 *SM);

void stream_merge_nd_2_192_term  (Stream_192 *S1, Stream_192 *S2, Stream_192 *SM);

void stream_merge_nd_2_256_term  (Stream_256 *S1, Stream_256 *S2, Stream_256 *S3);
```

```
void stream_merge_nd_2_320_term  (Stream_320 *S1, Stream_320 *S2, Stream_320 *SM);

void stream_merge_nd_2_512_term  (Stream_512 *S1, Stream_512 *S2, Stream_512 *SM);

void stream_merge_nd_2_1024_term (Stream_1024 *S1, Stream_1024 *S2, Stream_1024 *SM);
```

A.15.6.3 MxN Merges

```
void stream_merge_8_mxn        (Stream_8 *S0, Stream_8 *S1, Stream_8 *S2, int m, int n, int
                               cyc);

void stream_merge_16_mxn       (Stream_16 *S0, Stream_16 *S1, Stream_16 *S2, int m, int n, int
                               cyc);

void stream_merge_32_mxn       (Stream_32 *S0, Stream_32 *S1, Stream_32 *S2, int m, int n, int
                               cyc);

void stream_merge_64_mxn       (Stream_64 *S0, Stream_64 *S1, Stream_64 *S2, int m, int n, int
                               cyc);

void stream_merge_128_mxn      (Stream_128 *S0, Stream_128 *S1, Stream_128 *S2, int m, int n,
                               int cyc);

void stream_merge_192_mxn      (Stream_192 *S0, Stream_192 *S1, Stream_192 *S2, int m, int n,
                               int cyc);

void stream_merge_256_mxn      (Stream_256 *S0, Stream_256 *S1, Stream_256 *S2, int m, int n,
                               int cyc);

void stream_merge_320_mxn      (Stream_320 *S0, Stream_320 *S1, Stream_320 *S2, int m, int n,
                               int cyc);

void stream_merge_320_mxn_term (Stream_320 *S0, Stream_320 *S1, Stream_320 *S2, int m, int n);

void stream_merge_256_mxn_term (Stream_256 *S0, Stream_256 *S1, Stream_256 *S2, int m, int n);

void stream_merge_192_mxn_term (Stream_192 *S0, Stream_192 *S1, Stream_192 *S2, int m, int n);

void stream_merge_128_mxn_term (Stream_128 *S0, Stream_128 *S1, Stream_128 *S2, int m, int n);

void stream_merge_64_mxn_term  (Stream_64 *S0, Stream_64 *S1, Stream_64 *S2, int m, int n);

void stream_merge_32_mxn_term  (Stream_32 *S0, Stream_32 *S1, Stream_32 *S2, int m, int n);

void stream_merge_16_mxn_term  (Stream_16 *S0, Stream_16 *S1, Stream_16 *S2, int m, int n);

void stream_merge_8_mxn_term   (Stream_8 *S0, Stream_8 *S1, Stream_8 *S2, int m, int n);
```

A.15.6.4 Non-Deterministic MxN Merges

```
void stream_merge_64_mxn_nd_term (Stream_64 *S1, Stream_64 *S2, Stream_64 *S3, int m, int n);

void stream_merge_32_mxn_nd_term (Stream_32 *S1, Stream_32 *S2, Stream_32 *S3, int m, int n);

void stream_merge_16_mxn_nd_term (Stream_16 *S1, Stream_16 *S2, Stream_16 *S3, int m, int n);
```

```
void stream_merge_8_mxn_nd_term  (Stream_8  *S1, Stream_8  *S2, Stream_8  *S3, int m, int n);
```

A.15.6.5 VxV Merges

```
void stream_merge_64_vxv_var_term (Stream_64 *S1, Stream_32 *S1_count, Stream_64 *S2,
                            Stream_32 *S2_count, Stream_64 *SM, Stream_32 *SM_count);

void stream_merge_32_vxv_var_term (Stream_32 *S1, Stream_32 *S1_count, Stream_32 *S2,
                            Stream_32 *S2_count, Stream_32 *SM, Stream_32 *SM_count);

void stream_merge_16_vxv_var_term (Stream_16 *S1, Stream_32 *S1_count, Stream_16 *S2,
                            Stream_32 *S2_count, Stream_16 *SM, Stream_32 *SM_count);

void stream_merge_8_vxv_var_term  (Stream_8 *S1, Stream_32 *S1_count, Stream_8 *S2, Stream_32
                            *S2_count, Stream_8 *SM_count, Stream_32 *SM_count);
```

A.15.7 Stream Split

A.15.7.1 Round Robin Split

```
void stream_split_2_8        (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *Sout, int64_t num);

void stream_split_2_16       (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, int64_t
                             num)

void stream_split_2_32       (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, int64_t
                             num);

void stream_split_2_64       (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, int64_t
                             num);

void stream_split_3_8        (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, Stream_8
                             *S3_out, int64_t num);

void stream_split_3_16       (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, Stream_16
                             *S3_out, int64_t num);

void stream_split_3_32       (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, Stream_32
                             *S3_out, int64_t num);

void stream_split_3_64       (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, Stream_64
                             *S3_out, int64_t num);

void stream_split_4_8        (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, Stream_8
                             *S3_out, Stream_8 *S4_out, int64_t num);

void stream_split_4_16       (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, Stream_16
                             *S3_out, Stream_16 *S4_out, int64_t num);

void stream_split_4_32       (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, Stream_32
                             *S3_out, Stream_32 *S4_out, int64_t num);

void stream_split_4_64       (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, Stream_64
                             *S3_out, Stream_64 *S4_out, int64_t num);
```

```
void stream_split_5_8        (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, Stream_8
                             *S3_out, Stream_8 *S4_out, Stream_8 *S5_out, int64_t num);

void stream_split_5_16       (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, Stream_16
                             *S3_out, Stream_16 *S4_out, Stream_16 *S5_out, int64_t num);

void stream_split_5_32       (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, Stream_32
                             *S3_out, Stream_32 *S4_out, Stream_32 *S5_out, int64_t num);

void stream_split_5_64       (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, Stream_64
                             *S3_out, Stream_64 *S4_out, Stream_64 *S5_out, int64_t num);

void stream_split_2_8_term   (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out);

void stream_split_2_16_term  (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out);

void stream_split_2_32_term  (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out);

void stream_split_2_64_term  (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out);

void stream_split_3_8_term   (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, Stream_8
                             *S3_out);

void stream_split_3_16_term  (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, Stream_16
                             *S3_out);

void stream_split_3_32_term  (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, Stream_32
                             *S3_out);

void stream_split_3_64_term  (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, Stream_64
                             *S3_out);

void stream_split_4_8_term   (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, Stream_8
                             *S3_out, Stream_8 *S4_out);

void stream_split_4_16_term  (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, Stream_16
                             *S3_out, Stream_16 *S4_out);

void stream_split_4_32_term  (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, Stream_32
                             *S3_out, Stream_32 *S4_out);

void stream_split_4_64_term  (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, Stream_64
                             *S3_out, Stream_64 *S4_out);

void stream_split_5_8_term   (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, Stream_8
                             *S3_out, Stream_8 *S4_out, Stream_8 *S5_out);

void stream_split_5_16_term  (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, Stream_16
                             *S3_out, Stream_16 *S4_out, Stream_16 *S5_out);

void stream_split_5_32_term  (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, Stream_32
                             *S3_out, Stream_32 *S4_out, Stream_32 *S5_out);

void stream_split_5_64_term  (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, Stream_64
                             *S3_out, Stream_64 *S4_out, Stream_64 *S5_out);
```

A.15.7.2 MxN Split

```
void stream_split_8_mxn        (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, int m, int
                               n, int cyc);

void stream_split_16_mxn       (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, int m,
                               int n, int cyc);

void stream_split_32_mxn       (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, int m,
                               int n, int cyc);

void stream_split_64_mxn       (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, int m,
                               int n, int cyc);

void stream_split_128_mxn      (Stream_128 *S0, Stream_128 *S1_out, Stream_128 *S2_out, int m,
                               int n, int cyc);

void stream_split_192_mxn      (Stream_192 *S0, Stream_192 *S1_out, Stream_192 *S2_out, int m,
                               int n, int cyc);

void stream_split_256_mxn      (Stream_256 *S0, Stream_256 *S1_out, Stream_256 *S2_out, int m,
                               int n, int cyc);

void stream_split_320_mxn      (Stream_320 *S0, Stream_320 *S1_out, Stream_320 *S2_out, int m,
                               int n, int cyc);

void stream_split_320_mxn_term (Stream_320 *S0, Stream_320 *S1_out, Stream_320 *S2_out, int m,
                               int n);

void stream_split_256_mxn_term (Stream_256 *S0, Stream_256 *S1_out, Stream_256 *S2_out, int m,
                               int n);

void stream_split_192_mxn_term (Stream_192 *S0, Stream_192 *S1_out, Stream_192 *S2_out, int m,
                               int n);

void stream_split_128_mxn_term (Stream_128 *S0, Stream_128 *S1_out, Stream_128 *S2_out, int m,
                               int n);

void stream_split_64_mxn_term  (Stream_64 *S0, Stream_64 *S1_out, Stream_64 *S2_out, int m,
                               int n);

void stream_split_32_mxn_term  (Stream_32 *S0, Stream_32 *S1_out, Stream_32 *S2_out, int m,
                               int n);

void stream_split_16_mxn_term  (Stream_16 *S0, Stream_16 *S1_out, Stream_16 *S2_out, int m,
                               int n);

void stream_split_8_mxn_term   (Stream_8 *S0, Stream_8 *S1_out, Stream_8 *S2_out, int m, int
                               n);
```

A.15.7.3 MxN Cnt Split

```
void stream_split_64_mxn_cnts_term (Stream_64 *S, Stream_64 *S_counts, Stream_64 *S1_out,
                                    Stream_64 *S2_out);

void stream_split_32_mxn_cnts_term (Stream_32 *S, Stream_32 *S_counts, Stream_32 *S1_out,
                                    Stream_32 *S2_out);
```

A.15.8 Streaming Floating Point Accumulators

```
void stream_fp_accum_32_term                (Stream_32 *S, Stream_32 *Ssum, int num);

void stream_fp_accum_32_rr_term             (Stream_32 *S, Stream_32 *Ssum, int num);

void stream_fp_accum_strm_counts_32_term    (Stream_32 *S, Stream_32 *S_counts, Stream_32
                                    *Ssum);

void stream_fp_accum_strm_counts_32_rr_term (Stream_32 *S, Stream_32 *S_counts, Stream_32
                                    *Ssum);

void stream_fp_accum_strm_counts_64_term    (Stream_64 *S, Stream_64 *S_counts, Stream_64
                                    *Ssum);

void stream_fp_accum_strm_counts_64_rr_term (Stream_64 *S, Stream_64 *S_counts, Stream_64
                                    *Ssum);
```

A.15.9 Stream Width Conversion

A.15.9.1 Big-Endian Conversion

```
void stream_width_8to16     (Stream_8 *S1, Stream_16   *S2, int64_t count);

void stream_width_8to32     (Stream_8 *S1, Stream_32   *S2, int64_t count);

void stream_width_8to64     (Stream_8 *S1, Stream_64   *S2, int64_t count);

void stream_width_8to128    (Stream_8 *S1, Stream_128  *S2, int64_t count);

void stream_width_8to192    (Stream_8 *S1, Stream_192  *S2, int64_t count);

void stream_width_8to256    (Stream_8 *S1, Stream_256  *S2, int64_t count);

void stream_width_8to320    (Stream_8 *S1, Stream_320  *S2, int64_t count);

void stream_width_8to512    (Stream_8 *S1, Stream_512  *S2, int64_t count);

void stream_width_8to1024   (Stream_8 *S1, Stream_1024 *S2, int64_t count);

void stream_width_16to8     (Stream_16 *S1, Stream_8    *S2, int64_t count);

void stream_width_16to32    (Stream_16 *S1, Stream_32   *S2, int64_t count);

void stream_width_16to64    (Stream_16 *S1, Stream_64   *S2, int64_t count);

void stream_width_16to128   (Stream_16 *S1, Stream_128  *S2, int64_t count);
```

```
void stream_width_16to192   (Stream_16 *S1, Stream_192  *S2, int64_t count);
void stream_width_16to256   (Stream_16 *S1, Stream_256  *S2, int64_t count);
void stream_width_16to320   (Stream_16 *S1, Stream_320  *S2, int64_t count);
void stream_width_16to512   (Stream_16 *S1, Stream_512  *S2, int64_t count);
void stream_width_16to1024  (Stream_16 *S1, Stream_1024 *S2, int64_t count);
void stream_width_32to8     (Stream_32 *S1, Stream_8    *S2, int64_t count);
void stream_width_32to16    (Stream_32 *S1, Stream_16   *S2, int64_t count);
void stream_width_32to64    (Stream_32 *S1, Stream_64   *S2, int64_t count);
void stream_width_32to128   (Stream_32 *S1, Stream_128  *S2, int64_t count);
void stream_width_32to192   (Stream_32 *S1, Stream_192  *S2, int64_t count);
void stream_width_32to256   (Stream_32 *S1, Stream_256  *S2, int64_t count);
void stream_width_32to320   (Stream_32 *S1, Stream_320  *S2, int64_t count);
void stream_width_32to512   (Stream_32 *S1, Stream_512  *S2, int64_t count);
void stream_width_32to1024  (Stream_32 *S1, Stream_1024 *S2, int64_t count);
void stream_width_64to8     (Stream_64 *S1, Stream_8    *S2, int64_t count);
void stream_width_64to16    (Stream_64 *S1, Stream_16   *S2, int64_t count);
void stream_width_64to32    (Stream_64 *S1, Stream_32   *S2, int64_t count);
void stream_width_64to128   (Stream_64 *S1, Stream_128  *S2, int64_t count);
void stream_width_64to192   (Stream_64 *S1, Stream_192  *S2, int64_t count);
void stream_width_64to256   (Stream_64 *S1, Stream_256  *S2, int64_t count);
void stream_width_64to320   (Stream_64 *S1, Stream_320  *S2, int64_t count);
void stream_width_64to512   (Stream_64 *S1, Stream_512  *S2, int64_t count);
void stream_width_64to1024  (Stream_64 *S1, Stream_1024 *S2, int64_t count);
void stream_width_128to8    (Stream_128 *S1, Stream_8    *S2, int64_t count);
void stream_width_128to16   (Stream_128 *S1, Stream_16   *S2, int64_t count);
void stream_width_128to32   (Stream_128 *S1, Stream_32   *S2, int64_t count);
void stream_width_128to64   (Stream_128 *S1, Stream_64   *S2, int64_t count);
void stream_width_128to192  (Stream_128 *S1, Stream_192  *S2, int64_t count);
```

```
void stream_width_128to256  (Stream_128 *S1, Stream_256  *S2, int64_t count);
void stream_width_128to320  (Stream_128 *S1, Stream_320  *S2, int64_t count);
void stream_width_128to512  (Stream_128 *S1, Stream_512  *S2, int64_t count);
void stream_width_128to1024 (Stream_128 *S1, Stream_1024 *S2, int64_t count);
void stream_width_192to8    (Stream_192 *S1, Stream_8    *S2, int64_t count);
void stream_width_192to16   (Stream_192 *S1, Stream_16   *S2, int64_t count);
void stream_width_192to32   (Stream_192 *S1, Stream_32   *S2, int64_t count);
void stream_width_192to64   (Stream_192 *S1, Stream_64   *S2, int64_t count);
void stream_width_192to128  (Stream_192 *S1, Stream_128  *S2, int64_t count);
void stream_width_192to256  (Stream_192 *S1, Stream_256  *S2, int64_t count);
void stream_width_192to320  (Stream_192 *S1, Stream_320  *S2, int64_t count);
void stream_width_192to512  (Stream_192 *S1, Stream_512  *S2, int64_t count);
void stream_width_192to1024 (Stream_192 *S1, Stream_1024 *S2, int64_t count);
void stream_width_256to8    (Stream_256 *S1, Stream_8    *S2, int64_t count);
void stream_width_256to16   (Stream_256 *S1, Stream_16   *S2, int64_t count);
void stream_width_256to32   (Stream_256 *S1, Stream_32   *S2, int64_t count);
void stream_width_256to64   (Stream_256 *S1, Stream_64   *S2, int64_t count);
void stream_width_256to128  (Stream_256 *S1, Stream_128  *S2, int64_t count);
void stream_width_256to192  (Stream_256 *S1, Stream_192  *S2, int64_t count);
void stream_width_256to320  (Stream_256 *S1, Stream_320  *S2, int64_t count);
void stream_width_256to512  (Stream_256 *S1, Stream_512  *S2, int64_t count);
void stream_width_256to1024 (Stream_256 *S1, Stream_1024 *S2, int64_t count);
void stream_width_128to1024 (Stream_128 *S1, Stream_1024 *S2, int64_t count);
void stream_width_320to8    (Stream_320 *S1, Stream_8 *S2, int64_t count);
void stream_width_320to16   (Stream_320 *S1, Stream_16 *S2, int64_t count);
void stream_width_320to32   (Stream_320 *S1, Stream_32 *S2, int64_t count);
void stream_width_320to64   (Stream_320 *S1, Stream_64 *S2, int64_t count);
void stream_width_320to128  (Stream_320 *S1, Stream_128 *S2,int64_t count);
```

```
void stream_width_320to192  (Stream_320  *S1, Stream_192 *S2, int64_t count);
void stream_width_320to256  (Stream_320  *S1, Stream_256 *S2, int64_t count);
void stream_width_320to512  (Stream_320  *S1, Stream_512 *S2, int64_t count);
void stream_width_320to1024 (Stream_320  *S1, Stream_1024 *S2, int64_t count);
void stream_width_512to8    (Stream_512  *S1, Stream_8 *S2, int64_t count);
void stream_width_512to16   (Stream_512  *S1, Stream_16 *S2, int64_t count);
void stream_width_512to32   (Stream_512  *S1, Stream_32 *S2, int64_t count);
void stream_width_512to64   (Stream_512  *S1, Stream_64 *S2, int64_t count);
void stream_width_512to128  (Stream_512  *S1, Stream_128 *S2, int64_t count);
void stream_width_512to192  (Stream_512  *S1, Stream_192 *S2, int64_t count);
void stream_width_512to256  (Stream_512  *S1, Stream_256 *S2, int64_t count);
void stream_width_512to320  (Stream_512  *S1, Stream_320 *S2, int64_t count);
void stream_width_512to1024 (Stream_512  *S1, Stream_1024 *S2, int64_t count);
void stream_width_1024to8   (Stream_1024 *S1, Stream_8  *S2, int64_t count);
void stream_width_1024to16  (Stream_1024 *S1, Stream_16 *S2, int64_t count);
void stream_width_1024to32  (Stream_1024 *S1, Stream_32 *S2, int64_t count);
void stream_width_1024to64  (Stream_1024 *S1, Stream_64 *S2, int64_t count);
void stream_width_1024to128 (Stream_1024 *S1, Stream_128 *S2, int64_t count);
void stream_width_1024to192 (Stream_1024 *S1, Stream_192 *S2, int64_t count);
void stream_width_1024to256 (Stream_1024 *S1, Stream_256 *S2, int64_t count);
void stream_width_1024to320 (Stream_1024 *S1, Stream_320 *S2, int64_t count);
void stream_width_1024to512 (Stream_1024 *S1, Stream_512 *S2, int64_t count);
```

A.15.9.2 Little-Endian Conversion

```
void stream_width_8to16_le     (Stream_8  *S1, Stream_16 *S2, int64_t count);
void stream_width_8to32_le     (Stream_8  *S1, Stream_32 *S2, int64_t count);
void stream_width_8to64_le     (Stream_8  *S1, Stream_64 *S2, int64_t count);
void stream_width_8to128_le    (Stream_8 *S1, Stream_128 *S2, int64_t count);
void stream_width_8to192_le    (Stream_8 *S1, Stream_192 *S2, int64_t count);
```

```
void stream_width_8to256_le    (Stream_8 *S1, Stream_256 *S2, int64_t count);
void stream_width_8to320_le    (Stream_8 *S1, Stream_320 *S2, int64_t count);
void stream_width_8to512_le    (Stream_8 *S1, Stream_512  *S2, int64_t count);
void stream_width_8to1024_le   (Stream_8 *S1, Stream_1024 *S2, int64_t count);
void stream_width_16to8_le     (Stream_16 *S1, Stream_8 *S2, int64_t count);
void stream_width_16to32_le    (Stream_16 *S1, Stream_32   *S2, int64_t count);
void stream_width_16to64_le    (Stream_16 *S1, Stream_64   *S2, int64_t count);
void stream_width_16to128_le   (Stream_16 *S1, Stream_128 *S2, int64_t count);
void stream_width_16to192_le   (Stream_16 *S1, Stream_192 *S2, int64_t count);
void stream_width_16to256_le   (Stream_16 *S1, Stream_256 *S2, int64_t count);
void stream_width_16to320_le   (Stream_16 *S1, Stream_320 *S2, int64_t count);
void stream_width_16to512_le   (Stream_16 *S1, Stream_512 *S2, int64_t count);
void stream_width_16to1024_le  (Stream_16 *S1, Stream_1024 *S2, int64_t count);
void stream_width_32to8_le     (Stream_32 *S1, Stream_8 *S2, int64_t count);
void stream_width_32to16_le    (Stream_32 *S1, Stream_16 *S2, int64_t count);
void stream_width_32to64_le    (Stream_32 *S1, Stream_64 *S2, int64_t count);
void stream_width_32to128_le   (Stream_32 *S1, Stream_128 *S2, int64_t count);
void stream_width_32to192_le   (Stream_32 *S1, Stream_192 *S2, int64_t count);
void stream_width_32to256_le   (Stream_32 *S1, Stream_256 *S2, int64_t count);
void stream_width_32to320_le   (Stream_32 *S1, Stream_320 *S2, int64_t count);
void stream_width_32to512_le   (Stream_32 *S1, Stream_512 *S2, int64_t count);
void stream_width_32to1024_le  (Stream_32 *S1, Stream_1024 *S2, int64_t count);
void stream_width_64to8_le     (Stream_64 *S1, Stream_8 *S2, int64_t count);
void stream_width_64to16_le    (Stream_64 *S1, Stream_16 *S2, int64_t count);
void stream_width_64to32_le    (Stream_64 *S1, Stream_32 *S2, int64_t count);
void stream_width_64to128_le   (Stream_64 *S1, Stream_128 *S2, int64_t count);
void stream_width_64to192_le   (Stream_64 *S1, Stream_192 *S2, int64_t count);
void stream_width_64to256_le   (Stream_64 *S1, Stream_256 *S2, int64_t count);
```

```
void stream_width_64to320_le   (Stream_64 *S1, Stream_320 *S2, int64_t count);
void stream_width_64to512_le   (Stream_64 *S1, Stream_512 *S2, int64_t count);
void stream_width_64to1024_le  (Stream_64 *S1, Stream_1024 *S2, int64_t count);
void stream_width_128to8_le    (Stream_128 *S1, Stream_8 *S2, int64_t count);
void stream_width_128to16_le   (Stream_128 *S1, Stream_16 *S2, int64_t count);
void stream_width_128to32_le   (Stream_128 *S1, Stream_32 *S2, int64_t count);
void stream_width_128to64_le   (Stream_128 *S1, Stream_64 *S2, int64_t count);
void stream_width_128to192_le  (Stream_128 *S1, Stream_192 *S2, int64_t count);
void stream_width_128to256_le  (Stream_128 *S1, Stream_256 *S2, int64_t count);
void stream_width_128to320_le  (Stream_128 *S1, Stream_320 *S2, int64_t count);
void stream_width_128to512_le  (Stream_128 *S1, Stream_512 *S2, int64_t count);
void stream_width_128to1024_le (Stream_128 *S1, Stream_1024 *S2, int64_t count);
void stream_width_192to8_le    (Stream_192 *S1, Stream_8 *S2, int64_t count);
void stream_width_192to16_le   (Stream_192 *S1, Stream_16 *S2, int64_t count);
void stream_width_192to32_le   (Stream_192 *S1, Stream_32 *S2, int64_t count);
void stream_width_192to64_le   (Stream_192 *S1, Stream_64 *S2, int64_t count);
void stream_width_192to128_le  (Stream_192 *S1, Stream_128 *S2, int64_t count);
void stream_width_192to256_le  (Stream_192 *S1, Stream_256 *S2, int64_t count);
void stream_width_192to320_le  (Stream_192 *S1, Stream_320 *S2, int64_t count);
void stream_width_192to512_le  (Stream_192 *S1, Stream_512 *S2, int64_t count);
void stream_width_192to1024_le (Stream_192 *S1, Stream_1024 *S2, int64_t count);
void stream_width_256to8_le    (Stream_256 *S1, Stream_8 *S2, int64_t count);
void stream_width_256to16_le   (Stream_256 *S1, Stream_16 *S2, int64_t count);
void stream_width_256to32_le   (Stream_256 *S1, Stream_32 *S2, int64_t count);
void stream_width_256to64_le   (Stream_256 *S1, Stream_64 *S2, int64_t count);
void stream_width_256to128_le  (Stream_256 *S1, Stream_128 *S2, int64_t count);
void stream_width_256to192_le  (Stream_256 *S1, Stream_192 *S2, int64_t count);
void stream_width_256to320_le  (Stream_256 *S1, Stream_320 *S2, int64_t count);
```

```
void stream_width_256to512_le  (Stream_256 *S1, Stream_512 *S2, int64_t count);
void stream_width_256to1024_le (Stream_256 *S1, Stream_1024 *S2, int64_t count);
void stream_width_320to8_le    (Stream_320 *S1, Stream_8 *S2, int64_t count);
void stream_width_320to16_le   (Stream_320 *S1, Stream_16 *S2, int64_t count);
void stream_width_320to32_le   (Stream_320 *S1, Stream_32 *S2, int64_t count);
void stream_width_320to64_le   (Stream_320 *S1, Stream_64 *S2, int64_t count);
void stream_width_320to128_le  (Stream_320 *S1, Stream_128 *S2, int64_t count);
void stream_width_320to192_le  (Stream_320 *S1, Stream_192 *S2, int64_t count);
void stream_width_320to256_le  (Stream_320 *S1, Stream_256 *S2, int64_t count);
void stream_width_320to512_le  (Stream_320 *S1, Stream_512 *S2, int64_t count);
void stream_width_320to1024_le (Stream_320 *S1, Stream_1024 *S2, int64_t count);
void stream_width_512to8_le    (Stream_512 *S1, Stream_8 *S2, int64_t count);
void stream_width_512to16_le   (Stream_512 *S1, Stream_16 *S2, int64_t count);
void stream_width_512to32_le   (Stream_512 *S1, Stream_32 *S2, int64_t count);
void stream_width_512to64_le   (Stream_512 *S1, Stream_64 *S2, int64_t count);
void stream_width_512to128_le  (Stream_512 *S1, Stream_128 *S2, int64_t count);
void stream_width_512to192_le  (Stream_512 *S1, Stream_192 *S2, int64_t count);
void stream_width_512to256_le  (Stream_512 *S1, Stream_256 *S2, int64_t count);
void stream_width_512to320_le  (Stream_512 *S1, Stream_320 *S2, int64_t count);
void stream_width_512to1024_le (Stream_512 *S1, Stream_1024 *S2, int64_t count);
void stream_width_1024to8_le   (Stream_1024 *S1, Stream_8 *S2, int64_t count);
void stream_width_1024to16_le  (Stream_1024 *S1, Stream_16 *S2, int64_t count);
void stream_width_1024to32_le  (Stream_1024 *S1, Stream_32 *S2, int64_t count);
void stream_width_1024to64_le  (Stream_1024 *S1, Stream_64 *S2, int64_t count);
void stream_width_1024to128_le (Stream_1024 *S1, Stream_128 *S2, int64_t count);
void stream_width_1024to192_le (Stream_1024 *S1, Stream_192 *S2, int64_t count);
void stream_width_1024to256_le (Stream_1024 *S1, Stream_256 *S2, int64_t count);
void stream_width_1024to320_le (Stream_1024 *S1, Stream_320 *S2, int64_t count);
```

```
void stream_width_1024to512_le (Stream_1024 *S1, Stream_512 *S2, int64_t count);
```

A.15.9.3 Terminating Conversion – Big-Endian

```
void stream_width_8to16_term     (Stream_8    *S1, Stream_16   *S2);
void stream_width_8to32_term     (Stream_8    *S1, Stream_32   *S2);
void stream_width_8to64_term     (Stream_8    *S1, Stream_64   *S2);
void stream_width_8to128_term    (Stream_8    *S1, Stream_128  *S2);
void stream_width_8to192_term    (Stream_8    *S1, Stream_192  *S2);
void stream_width_8to256_term    (Stream_8    *S1, Stream_256  *S2);
void stream_width_8to320_term    (Stream_8    *S1, Stream_320  *S2);
void stream_width_8to512_term    (Stream_8    *S1, Stream_512  *S2);
void stream_width_8to1024_term   (Stream_8    *S1, Stream_1024 *S2);
void stream_width_16to8_term     (Stream_16   *S1, Stream_8    *S2);
void stream_width_16to32_term    (Stream_16   *S1, Stream_32   *S2);
void stream_width_16to64_term    (Stream_16   *S1, Stream_64   *S2);
void stream_width_16to128_term   (Stream_16   *S1, Stream_128  *S2);
void stream_width_16to192_term   (Stream_16   *S1, Stream_192  *S2);
void stream_width_16to256_term   (Stream_16   *S1, Stream_256  *S2);
void stream_width_16to320_term   (Stream_16   *S1, Stream_320  *S2);
void stream_width_16to512_term   (Stream_16   *S1, Stream_512  *S2);
void stream_width_16to1024_term  (Stream_16   *S1, Stream_1024 *S2);
void stream_width_32to8_term     (Stream_32   *S1, Stream_8    *S2);
void stream_width_32to16_term    (Stream_32   *S1, Stream_16   *S2);
void stream_width_32to64_term    (Stream_32   *S1, Stream_64   *S2);
void stream_width_32to128_term   (Stream_32   *S1, Stream_128  *S2);
void stream_width_32to192_term   (Stream_32   *S1, Stream_192  *S2);
void stream_width_32to256_term   (Stream_32   *S1, Stream_256  *S2);
void stream_width_32to320_term   (Stream_32   *S1, Stream_320  *S2);
void stream_width_32to512_term   (Stream_32   *S1, Stream_512  *S2);
```

```
void stream_width_32to1024_term  (Stream_32   *S1, Stream_1024 *S2);

void stream_width_64to8_term     (Stream_64   *S1, Stream_8    *S2);

void stream_width_64to16_term    (Stream_64   *S1, Stream_16   *S2);

void stream_width_64to32_term    (Stream_64   *S1, Stream_32   *S2);

void stream_width_64to128_term   (Stream_64   *S1, Stream_128  *S2);

void stream_width_64to192_term   (Stream_64   *S1, Stream_192  *S2);

void stream_width_64to256_term   (Stream_64   *S1, Stream_256  *S2);

void stream_width_64to320_term   (Stream_64   *S1, Stream_320  *S2);

void stream_width_64to512_term   (Stream_64   *S1, Stream_512  *S2);

void stream_width_64to1024_term  (Stream_64   *S1, Stream_1024 *S2);

void stream_width_128to8_term    (Stream_128  *S1, Stream_8    *S2);

void stream_width_128to16_term   (Stream_128  *S1, Stream_16   *S2);

void stream_width_128to32_term   (Stream_128  *S1, Stream_32   *S2);

void stream_width_128to64_term   (Stream_128  *S1, Stream_64   *S2);

void stream_width_128to192_term  (Stream_128  *S1, Stream_192  *S2);

void stream_width_128to256_term  (Stream_128  *S1, Stream_256  *S2);

void stream_width_128to320_term  (Stream_128  *S1, Stream_320  *S2);

void stream_width_128to512_term  (Stream_128  *S1, Stream_512  *S2);

void stream_width_128to1024_term (Stream_128  *S1, Stream_1024 *S2);

void stream_width_192to8_term    (Stream_192  *S1, Stream_8    *S2);

void stream_width_192to16_term   (Stream_192  *S1, Stream_16   *S2);

void stream_width_192to32_term   (Stream_192  *S1, Stream_32   *S2);

void stream_width_192to64_term   (Stream_192  *S1, Stream_64   *S2);

void stream_width_192to128_term  (Stream_192  *S1, Stream_128  *S2);

void stream_width_192to256_term  (Stream_192  *S1, Stream_256  *S2);

void stream_width_192to320_term  (Stream_192  *S1, Stream_320  *S2);

void stream_width_192to512_term  (Stream_192  *S1, Stream_512  *S2);

void stream_width_192to1024_term (Stream_192  *S1, Stream_1024 *S2);
```

```
void stream_width_256to8_term    (Stream_256  *S1, Stream_8    *S2);
void stream_width_256to16_term   (Stream_256  *S1, Stream_16   *S2);
void stream_width_256to32_term   (Stream_256  *S1, Stream_32   *S2);
void stream_width_256to64_term   (Stream_256  *S1, Stream_64   *S2);
void stream_width_256to128_term  (Stream_256  *S1, Stream_128  *S2);
void stream_width_256to192_term  (Stream_256  *S1, Stream_192  *S2);
void stream_width_256to320_term  (Stream_256  *S1, Stream_320  *S2);
void stream_width_256to512_term  (Stream_256  *S1, Stream_512  *S2);
void stream_width_256to1024_term (Stream_256  *S1, Stream_1024 *S2);
void stream_width_128to1024_term (Stream_128  *S1, Stream_1024 *S2);
void stream_width_320to8_term    (Stream_320  *S1, Stream_8    *S2);
void stream_width_320to16_term   (Stream_320  *S1, Stream_16   *S2);
void stream_width_320to32_term   (Stream_320  *S1, Stream_32   *S2);
void stream_width_320to64_term   (Stream_320  *S1, Stream_64   *S2);
void stream_width_320to128_term  (Stream_320  *S1, Stream_128  *S2);
void stream_width_320to192_term  (Stream_320  *S1, Stream_192  *S2);
void stream_width_320to256_term  (Stream_320  *S1, Stream_256  *S2);
void stream_width_320to512_term  (Stream_320  *S1, Stream_512  *S2);
void stream_width_320to1024_term (Stream_320  *S1, Stream_1024 *S2);
void stream_width_512to8_term    (Stream_512  *S1, Stream_8    *S2);
void stream_width_512to16_term   (Stream_512  *S1, Stream_16   *S2);
void stream_width_512to32_term   (Stream_512  *S1, Stream_32   *S2);
void stream_width_512to64_term   (Stream_512  *S1, Stream_64   *S2);
void stream_width_512to128_term  (Stream_512  *S1, Stream_128  *S2);
void stream_width_512to192_term  (Stream_512  *S1, Stream_192  *S2);
void stream_width_512to256_term  (Stream_512  *S1, Stream_256  *S2);
void stream_width_512to320_term  (Stream_512  *S1, Stream_320  *S2);
void stream_width_512to1024_term (Stream_512  *S1, Stream_1024 *S2);
```

```
void stream_width_1024to8_term   (Stream_1024 *S1, Stream_8    *S2);

void stream_width_1024to16_term  (Stream_1024 *S1, Stream_16   *S2);

void stream_width_1024to32_term  (Stream_1024 *S1, Stream_32   *S2);

void stream_width_1024to64_term  (Stream_1024 *S1, Stream_64   *S2);

void stream_width_1024to128_term (Stream_1024 *S1, Stream_128  *S2);

void stream_width_1024to192_term (Stream_1024 *S1, Stream_192  *S2);

void stream_width_1024to256_term (Stream_1024 *S1, Stream_256  *S2);

void stream_width_1024to320_term (Stream_1024 *S1, Stream_320  *S2);

void stream_width_1024to512_term (Stream_1024 *S1, Stream_512  *S2);
```

A.15.9.4 Terminating Conversion – Little-Endian

```
void stream_width_8to16_le_term     (Stream_8    *S1, Stream_16   *S2);

void stream_width_8to32_le_term     (Stream_8    *S1, Stream_32   *S2);

void stream_width_8to64_le_term     (Stream_8    *S1, Stream_64   *S2);

void stream_width_8to128_le_term    (Stream_8    *S1, Stream_128  *S2);

void stream_width_8to192_le_term    (Stream_8    *S1, Stream_192  *S2);

void stream_width_8to256_le_term    (Stream_8    *S1, Stream_256  *S2);

void stream_width_8to320_le_term    (Stream_8    *S1, Stream_320  *S2);

void stream_width_8to512_le_term    (Stream_8    *S1, Stream_512  *S2);

void stream_width_8to1024_le_term   (Stream_8    *S1, Stream_1024 *S2);

void stream_width_16to8_le_term     (Stream_16   *S1, Stream_8    *S2);

void stream_width_16to32_le_term    (Stream_16   *S1, Stream_32   *S2);

void stream_width_16to64_le_term    (Stream_16   *S1, Stream_64   *S2);

void stream_width_16to128_le_term   (Stream_16   *S1, Stream_128  *S2);

void stream_width_16to192_le_term   (Stream_16   *S1, Stream_192  *S2);

void stream_width_16to256_le_term   (Stream_16   *S1, Stream_256  *S2);

void stream_width_16to320_le_term   (Stream_16   *S1, Stream_320  *S2);

void stream_width_16to512_le_term   (Stream_16   *S1, Stream_512  *S2);

void stream_width_16to1024_le_term  (Stream_16   *S1, Stream_1024 *S2);
```

```
void stream_width_32to8_le_term     (Stream_32   *S1, Stream_8    *S2);
void stream_width_32to16_le_term    (Stream_32   *S1, Stream_16   *S2);
void stream_width_32to64_le_term    (Stream_32   *S1, Stream_64   *S2);
void stream_width_32to128_le_term   (Stream_32   *S1, Stream_128  *S2);
void stream_width_32to192_le_term   (Stream_32   *S1, Stream_192  *S2);
void stream_width_32to256_le_term   (Stream_32   *S1, Stream_256  *S2);
void stream_width_32to320_le_term   (Stream_32   *S1, Stream_320  *S2);
void stream_width_32to512_le_term   (Stream_32   *S1, Stream_512  *S2);
void stream_width_32to1024_le_term  (Stream_32   *S1, Stream_1024 *S2);
void stream_width_64to8_le_term     (Stream_64   *S1, Stream_8    *S2);
void stream_width_64to16_le_term    (Stream_64   *S1, Stream_16   *S2);
void stream_width_64to32_le_term    (Stream_64   *S1, Stream_32   *S2);
void stream_width_64to128_le_term   (Stream_64   *S1, Stream_128  *S2);
void stream_width_64to192_le_term   (Stream_64   *S1, Stream_192  *S2);
void stream_width_64to256_le_term   (Stream_64   *S1, Stream_256  *S2);
void stream_width_64to320_le_term   (Stream_64   *S1, Stream_320  *S2);
void stream_width_64to512_le_term   (Stream_64   *S1, Stream_512  *S2);
void stream_width_64to1024_le_term  (Stream_64   *S1, Stream_1024 *S2);
void stream_width_128to8_le_term    (Stream_128  *S1, Stream_8    *S2);
void stream_width_128to16_le_term   (Stream_128  *S1, Stream_16   *S2);
void stream_width_128to32_le_term   (Stream_128  *S1, Stream_32   *S2);
void stream_width_128to64_le_term   (Stream_128  *S1, Stream_64   *S2);
void stream_width_128to192_le_term  (Stream_128  *S1, Stream_192  *S2);
void stream_width_128to256_le_term  (Stream_128  *S1, Stream_256  *S2);
void stream_width_128to320_le_term  (Stream_128  *S1, Stream_320  *S2);
void stream_width_128to512_le_term  (Stream_128  *S1, Stream_512  *S2);
void stream_width_128to1024_le_term (Stream_128  *S1, Stream_1024 *S2);
void stream_width_192to8_le_term    (Stream_192  *S1, Stream_8    *S2);
```

```
void stream_width_192to16_le_term   (Stream_192  *S1, Stream_16   *S2);
void stream_width_192to32_le_term   (Stream_192  *S1, Stream_32   *S2);
void stream_width_192to64_le_term   (Stream_192  *S1, Stream_64   *S2);
void stream_width_192to128_le_term  (Stream_192  *S1, Stream_128  *S2);
void stream_width_192to256_le_term  (Stream_192  *S1, Stream_256  *S2);
void stream_width_192to320_le_term  (Stream_192  *S1, Stream_320  *S2);
void stream_width_192to512_le_term  (Stream_192  *S1, Stream_512  *S2);
void stream_width_192to1024_le_term (Stream_192  *S1, Stream_1024 *S2);
void stream_width_256to8_le_term    (Stream_256  *S1, Stream_8    *S2);
void stream_width_256to16_le_term   (Stream_256  *S1, Stream_16   *S2);
void stream_width_256to32_le_term   (Stream_256  *S1, Stream_32   *S2);
void stream_width_256to64_le_term   (Stream_256  *S1, Stream_64   *S2);
void stream_width_256to128_le_term  (Stream_256  *S1, Stream_128  *S2);
void stream_width_256to192_le_term  (Stream_256  *S1, Stream_192  *S2);
void stream_width_256to320_le_term  (Stream_256  *S1, Stream_320  *S2);
void stream_width_256to512_le_term  (Stream_256  *S1, Stream_512  *S2);
void stream_width_256to1024_le_term (Stream_256  *S1, Stream_1024 *S2);
void stream_width_128to1024_le_term (Stream_128  *S1, Stream_1024 *S2);
void stream_width_320to8_le_term    (Stream_320  *S1, Stream_8    *S2);
void stream_width_320to16_le_term   (Stream_320  *S1, Stream_16   *S2);
void stream_width_320to32_le_term   (Stream_320  *S1, Stream_32   *S2);
void stream_width_320to64_le_term   (Stream_320  *S1, Stream_64   *S2);
void stream_width_320to128_le_term  (Stream_320  *S1, Stream_128  *S2);
void stream_width_320to192_le_term  (Stream_320  *S1, Stream_192  *S2);
void stream_width_320to256_le_term  (Stream_320  *S1, Stream_256  *S2);
void stream_width_320to512_le_term  (Stream_320  *S1, Stream_512  *S2);
void stream_width_320to1024_le_term (Stream_320  *S1, Stream_1024 *S2);
void stream_width_512to8_le_term    (Stream_512  *S1, Stream_8    *S2);
```

```
void stream_width_512to16_le_term   (Stream_512  *S1, Stream_16   *S2);

void stream_width_512to32_le_term   (Stream_512  *S1, Stream_32   *S2);

void stream_width_512to64_le_term   (Stream_512  *S1, Stream_64   *S2);

void stream_width_512to128_le_term  (Stream_512  *S1, Stream_128  *S2);

void stream_width_512to192_le_term  (Stream_512  *S1, Stream_192  *S2);

void stream_width_512to256_le_term  (Stream_512  *S1, Stream_256  *S2);

void stream_width_512to320_le_term  (Stream_512  *S1, Stream_320  *S2);

void stream_width_512to1024_le_term (Stream_512  *S1, Stream_1024 *S2);

void stream_width_1024to8_le_term   (Stream_1024 *S1, Stream_8    *S2);

void stream_width_1024to16_le_term  (Stream_1024 *S1, Stream_16   *S2);

void stream_width_1024to32_le_term  (Stream_1024 *S1, Stream_32   *S2);

void stream_width_1024to64_le_term  (Stream_1024 *S1, Stream_64   *S2);

void stream_width_1024to128_le_term (Stream_1024 *S1, Stream_128  *S2);

void stream_width_1024to192_le_term (Stream_1024 *S1, Stream_192  *S2);

void stream_width_1024to256_le_term (Stream_1024 *S1, Stream_256  *S2);

void stream_width_1024to320_le_term (Stream_1024 *S1, Stream_320  *S2);

void stream_width_1024to512_le_term (Stream_1024 *S1, Stream_512  *S2);
```

A.16 Vector Stream Functions

A.16.1 Producer Functions

The following functions send values to a vector stream dependent on an enable bit:

```
void put_vec_stream_8                (Vec_Stream_8* S, int8_t val, int enable);

void put_vec_stream_16               (Vec_Stream_16* S, int16_t val, int enable);

void put_vec_stream_32               (Vec_Stream_32* S, int32_t val, int enable);

void put_vec_stream_64               (Vec_Stream_64* S, int64_t val, int enable);

void put_vec_stream_128              (Vec_Stream_128* S, int64_t val1, int64_t val2, int
                                enable);

void put_vec_stream_256              (Vec_Stream_256* S, int64_t val1, int64_t val2, int64_t
                                val3, int64_t val4, int enable);
```

```
void put_vec_stream_512              (Vec_Stream_512* S, int64_t val1, int64_t val2, int64_t
                                val3, int64_t val4, int64_t val5, int64_t val6, int64_t val7,
                                int64_t val8, int enable);

void put_vec_stream_1024             (Vec_Stream_1024* S, int64_t val1, int64_t val2, int64_t
                                val3, int64_t val4, int64_t val5, int64_t val6, int64_t val7,
                                int64_t val8, int64_t val9, int64_t val10, int64_t val11, int64_t
                                val12, int64_t val13, int64_t val14, int64_t val15, int64_t
                                val16, int enable);

void put_vec_stream_flt_32           (Vec_Stream_32* S, float val, int enable);

void put_vec_stream_dbl_64           (Vec_Stream_64* S, double val, int enable);

void put_vec_stream_dbl_128          (Vec_Stream_128* S, double val1, double val2, int
                                enable);

void put_vec_stream_dbl_256          (Vec_Stream_256* S, double val1, double val2, double
                                val3, double val4, int enable);

void put_vec_stream_dbl_512          (Vec_Stream_512* S, double val1, double val2, double
                                val3, double val4, double val5, double val6, double val7, double
                                val8, int enable);

void put_vec_stream_dbl_1024         (Vec_Stream_1024* S, double val1, double val2, double
                                val3, double val4, double val5, double val6, double val7, double
                                val8, double val9, double val10, double val11, double val12,
                                double val13, double val14, double val15, double val16, int
                                enable);

void put_vec_stream_8_header         (Vec_Stream_8* S, int8_t val);

void put_vec_stream_16_header        (Vec_Stream_16* S, int16_t val);

void put_vec_stream_32_header        (Vec_Stream_32* S, int32_t val);

void put_vec_stream_64_header        (Vec_Stream_64* S, int64_t val);

void put_vec_stream_64_header_2val   (Vec_Stream_64* S, int64_t header, int64_t val);

void put_vec_stream_128_header       (Vec_Stream_128* S, int64_t val1, int64_t val2);

void put_vec_stream_256_header       (Vec_Stream_256* S, int64_t val1, int64_t val2, int64_t
                                val3, int64_t val4);

void put_vec_stream_512_header       (Vec_Stream_512* S, int64_t val1, int64_t val2, int64_t
                                val3, int64_t val4, int64_t val5, int64_t val6, int64_t val7,
                                int64_t val8);

void put_vec_stream_1024_header      (Vec_Stream_1024* S, int64_t val1, int64_t val2, int64_t
                                val3, int64_t val4, int64_t val5, int64_t val6, int64_t val7,
                                int64_t val8, int64_t val9, int64_t val10, int64_t val11, int64_t
                                val12, int64_t val13, int64_t val14, int64_t val15, int64_t
                                val16);

void put_vec_stream_flt_32_header    (Vec_Stream_32* S, float val);
```

```
void put_vec_stream_dbl_64_header     (Vec_Stream_64* S, double val);

voidput_vec_stream_dbl_64_header_2val (Vec_Stream_64* S, double header, double val);

void put_vec_stream_dbl_128_header    (Vec_Stream_128* S, double val1, double val2);

void put_vec_stream_dbl_256_header    (Vec_Stream_256* S, double val1, double val2, double
                            val3, double val4);

void put_vec_stream_dbl_512_header    (Vec_Stream_512* S, double val1, double val2, double
                            val3, double val4, double val5, double val6, double val7, double
                            val8);

void put_vec_stream_dbl_1024_header   (Vec_Stream_1024* S, double val1, double val2, double
                            val3, double val4, double val5, double val6, double val7, double
                            val8, double val9, double val10, double val11, double val12,
                            double val13, double val14, double val15, double val16);

void put_vec_stream_8_tail            (Vec_Stream_8* S, int8_t val);

void put_vec_stream_16_tail           (Vec_Stream_16* S, int16_t val);

void put_vec_stream_32_tail           (Vec_Stream_32* S, int32_t val);

void put_vec_stream_64_tail           (Vec_Stream_64* S, int64_t val);

void put_vec_stream_128_tail          (Vec_Stream_128* S, int64_t val1, int64_t val2);

void put_vec_stream_256_tail          (Vec_Stream_256* S, int64_t val1, int64_t val2, int64_t
                            val3, int64_t val4);

void put_vec_stream_512_tail          (Vec_Stream_512* S, int64_t val1, int64_t val2, int64_t
                            val3, int64_t val4, int64_t val5, int64_t val6, int64_t val7,
                            int64_t val8);

void put_vec_stream_1024_tail         (Vec_Stream_1024* S, int64_t val1, int64_t val2, int64_t
                            val3, int64_t val4, int64_t val5, int64_t val6, int64_t val7,
                            int64_t val8, int64_t val9, int64_t val10, int64_t val11, int64_t
                            val12, int64_t val13, int64_t val14, int64_t val15, int64_t
                            val16);

void put_vec_stream_flt_32_header     (Vec_Stream_32* S, float val);

void put_vec_stream_dbl_64_header     (Vec_Stream_64* S, double val);

void put_vec_stream_dbl_128_header    (Vec_Stream_128* S, double val1, double val2);

void put_vec_stream_dbl_256_header    (Vec_Stream_256* S, double val1, double val2, double
                            val3, double val4);

void put_vec_stream_dbl_512_header    (Vec_Stream_512* S, double val1, double val2, double
                            val3, double val4, double val5, double val6, double val7, double
                            val8);
```

```
void put_vec_stream_dbl_1024_header   (Vec_Stream_1024* S, double val1, double val2, double
                              val3, double val4, double val5, double val6, double val7, double
                              val8, double val9, double val10, double val11, double val12,
                              double val13, double val14, double val15, double val16);
```

A.16.2 Consumer Functions

These functions retrieve values from a stream:

```
void get_vec_stream_8                 (Vec_Stream_8* S, int8_t* val);

void get_vec_stream_16                (Vec_Stream_16* S, int16_t* val);

void get_vec_stream_32                (Vec_Stream_32* S, int32_t* val);

void get_vec_stream_64                (Vec_Stream_64* S, int64_t* val);

void get_vec_stream_128               (Vec_Stream_128* S, void* val1, void* val2);

void get_vec_stream_256               (Vec_Stream_256* S, void* val1, void* val2, void* val3,
                              void* val4);

void get_vec_stream_512               (Vec_Stream_512* S, void* val1, void* val2, void* val3,
                              void* val4, void* val5, void* val6, void* val7, void* val8);

void get_vec_stream_1024              (Vec_Stream_1024* S, void* val1, void*val2, void* val3,
                              void* val4, void* val5, void* val6, void* val7, void* val8, void*
                              val9, void* val10, void* val11, void* val12, void* val13,
                              void*val14, void* val15, void* val16);

void get_vec_stream_flt_32            (Vec_Stream_32* S, float* val);

void get_vec_stream_dbl_64            (Vec_Stream_64* S, double* val);

void get_vec_stream_dbl_128           (Vec_Stream_128* S, double* val1, double* val2);

void get_vec_stream_dbl_256           (Vec_Stream_256* S, double* val1, double* val2, double*
                              val3, double* val4);

void get_vec_stream_dbl_512           (Vec_Stream_512* S, double* val1, double* val2, double*
                              val3, double* val4, double* val5, double* val6, double* val7,
                              double* val8);

void get_vec_stream_dbl_1024          (Vec_Stream_512* S, double* val1, double* val2, double*
                              val3, double* val4, double*, double*, double*, double*, double*,
                              double*, double*, double*, double*, double*, double*, double*);

void get_vec_stream_8_header          (Vec_Stream_8* S, int8_t* val);

void get_vec_stream_16_header         (Vec_Stream_16* S, int16_t* val);

void get_vec_stream_32_header         (Vec_Stream_32* S, int32_t* val);

void get_vec_stream_64_header         (Vec_Stream_64* S, int64_t* val);

void get_vec_stream_64_header_2val    (Vec_Stream_64* S, int64_t* header, int64_t* val);
```

```
void get_vec_stream_128_header         (Vec_Stream_128* S, void* val1, void* val2);

void get_vec_stream_256_header         (Vec_Stream_256* S, void* val1, void* val2, void* val3,
                              void* val4);

void get_vec_stream_512_header         (Vec_Stream_512* S, void* val1, void* val2, void* val3,
                              void* val4, void* val5, void* val6, void* val7, void* val8);

void get_vec_stream_1024_header        (Vec_Stream_1024* S, void* val1, void*val2, void* val3,
                              void* val4, void* val5, void* val6, void* val7, void* val8, void*
                              val9, void* val10, void* val11, void* val12, void* val13,
                              void*val14, void* val15, void* val16);

void get_vec_stream_flt_32_header      (Vec_Stream_32* S, float* val);

void get_vec_stream_dbl_64_header      (Vec_Stream_64* S, double* val);

void get_vec_stream_dbl_64_header_2val (Vec_Stream_64* S, double* header, double* val);

void get_vec_stream_dbl_128_header     (Vec_Stream_128* S, double* val1, double* val2);

void get_vec_stream_dbl_256_header     (Vec_Stream_256* S, double* val1, double* val2, double*
                              val3, double* val4);

void get_vec_stream_dbl_512_header     (Vec_Stream_512* S, double* val1, double* val2, double*
                              val3, double* val4, double* val5, double* val6, double* val7,
                              double* val8);

void get_vec_stream_dbl_1024_header    (Vec_Stream_1024512* S, double* val1, double* val2,
                              double* val3, double* val4, double*, double*, double*, double*,
                              double*, double*, double*, double*, double*, double*, double*,
                              double*);

void get_vec_stream_8_tail             (Vec_Stream_8* S, int8_t* val);

void get_vec_stream_16_tail            (Vec_Stream_16* S, int16_t* val);

void get_vec_stream_32_tail            (Vec_Stream_32* S, int32_t* val);

void get_vec_stream_64_tail            (Vec_Stream_64* S, int64_t* val);

void get_vec_stream_128_tail           (Vec_Stream_128* S, void* val1, void* val2);

void get_vec_stream_256_tail           (Vec_Stream_256* S, void* val1, void* val2, void* val3,
                              void* val4);

void get_vec_stream_512_tail           (Vec_Stream_512* S, void* val1, void* val2, void* val3,
                              void* val4, void* val5, void* val6, void* val7, void* val8);

void get_vec_stream_1024_tail          (Vec_Stream_1024* S, void* val1, void*val2, void* val3,
                              void* val4, void* val5, void* val6, void* val7, void* val8, void*
                              val9, void* val10, void* val11, void* val12, void* val13,
                              void*val14, void* val15, void* val16);

void get_vec_stream_flt_32_tail        (Vec_Stream_32* S, float* val);
```

```
void get_vec_stream_dbl_64_tail        (Vec_Stream_64* S, double* val);

void get_vec_stream_dbl_128_tail       (Vec_Stream_128* S, double* val1, double* val2);

void get_vec_stream_dbl_256_tail       (Vec_Stream_256* S, double* val1, double* val2, double*
                                val3, double* val4);

void get_vec_stream_dbl_512_tail       (Vec_Stream_512* S, double* val1, double* val2, double*
                                val3, double* val4, double* val5, double* val6, double* val7,
                                double* val8);

void get_vec_stream_dbl_1024_tail      (Vec_Stream_1024512* S, double* val1, double* val2,
                                double* val3, double* val4, double*, double*, double*, double*,
                                double*, double*, double*, double*, double*, double*, double*,
                                double*);
```

A.16.3 Termination

```
void vec_stream_8_term    (Vec_Stream_8 *S);

void vec_stream_16_term   (Vec_Stream_16 *S);

void vec_stream_32_term   (Vec_Stream_32 *S);

void vec_stream_64_term   (VecStream_64 *S);

void vec_stream_128_term  (Vec_Stream_128 *S);

void vec_stream_256_term  (Vec_Stream_256 *S);

void vec_stream_512_term  (Vec_Stream_512 *S);

void vec_stream_1024_term (Vec_Stream_1024 *S);
```

A.16.4 Testing for Vector Stream Termination

```
int all_vec_streams_active    (void);

int is_vec_stream_8_active    (Vec_Stream_8 *S);

int is_vec_stream_16_active   (Vec_Stream_16 *S);

int is_vec_stream_32_active   (Vec_Stream_32 *S);

int is_vec_stream_64_active   (Vec_Stream_64 *S);

int is_vec_stream_128_active  (Vec_Stream_128 *S);

int is_vec_stream_256_active  (Vec_Stream_256 *S);

int is_vec_stream_512_active  (Vec_Stream_512 *S);

int is_vec_stream_1024_active (Vec_Stream_1024 *S);
```

A.16.5 Vector Stream Width Conversions

A.16.5.1 Terminating Conversion – Big-Endian

```
void vec_stream_width_8to64_term (Vec_Stream_8 *S1, Vec_Stream_64 *S2);

void vec_stream_width_16to64_term (Vec_Stream_16 *S1, Vec_Stream_64 *S2);

void vec_stream_width_32to64_term (Vec_Stream_32 *S1, Vec_Stream_64 *S2);

void vec_stream_width_64to8_term (Vec_Stream_64 *S1, Vec_Stream_8 *S2);

void vec_stream_width_64to16_term (Vec_Stream_64 *S1, Vec_Stream_16 *S2);

void vec_stream_width_64to32_term (Vec_Stream_64 *S1, Vec_Stream_32 *S2);

void vec_stream_width_64to256_term (Vec_Stream_64 *S1, Vec_Stream_256 *S2);

void vec_stream_width_256to64_term (Vec_stream_256 *S1, Vec_Stream_64 *S2);
```

A.16.5.2 Terminating Conversions – Little-Endian

```
void vec_stream_width_8to64_le_term (Vec_Stream_8 *S1, Vec_Stream_64 *S2);

void vec_stream_width_16to64_le_term (Vec_Stream_16 *S1, Vec_Stream_64 *S2);

void vec_stream_width_32to64_le_term (Vec_Stream_32 *S1, Vec_Stream_64 *S2);

void vec_stream_width_64to8_le_term (Vec_Stream_64 *S1, Vec_Stream_8 *S2);

void vec_stream_width_64to16_le_term (Vec_Stream_64 *S1, Vec_Stream_16 *S2);

void vec_stream_width_64to32_le_term (Vec_Stream_64 *S1, Vec_Stream_32 *S2);

void vec_stream_width_64to256_le_term (Vec_Stream_64 *S1, Vec_Stream_256 *S2);

void vec_stream_width_256to64_le_term (Vec_stream_256 *S1, Vec_Stream_64 *S2);
```

A.16.6 Vector Stream Merges

A.16.6.1 Terminating Deterministic Merges

```
void vec_stream_merge_2_64_term  (Vec_Stream_64 *S1, Vec_Stream_64 *S2, Vec_Stream_64 *SM);

void vec_stream_merge_2_256_term (Vec_Stream_256 *S1, Vec_Stream_256 *S2, Vec_Stream_256 *SM);
```

A.16.6.2 Terminating Non-Deterministic Merges

```
void vec_stream_merge_nd_2_64_term   (Vec_Stream_64 *S1, Vec_Stream_64 *S2, Vec_Stream_64
                                *SM);

void  vec_stream_merge_nd_2_256_term (Vec_Stream_256 *S1, Vec_Stream_256 *S2, Vec_Stream_256
                                *SM);
```

A.17 Carte Ethernet IPC

```
void carte_accept (uint32_t listening_sockfd, uint8_t *sin_family, uint16_t *sin_port,
                           uint32_t *sin_addr, uint32_t *connected_sockfd, uint32_t *res);

void carte_bind (uint32_t sockfd, uint8_t sin_family, uint16_t sin_port, uint32_t sin_addr,
                           uint32_t *res);

void carte_getipaddr (uint32_t data_if, uint32_t *sin_addr, uint32_t *res);

void carte_getifmask (uint8_t *ifmask, uint32_t *res);

void getpeername (uint32_t sockfd, uint8_t *sin_family, uint16_t *sin_port, uint32_t
                           *sin_addr, uint32_t *res);

void carte_getsockname (uint32_t sockfd, uint8_t *sin_family, uint16_t *sin_port, uint32_t
                           *sin_addr, uint32_t *res);

void carte_listen (uint32_t sockfd, uint32_t backlog, uint32_t *res);

void carte_setsockopt (uint32_t sockfd, uint32_t level, uint32_t option, uint32_t val,
                           uint32_t *res);

void carte_socket (uint8_t sin_dataif, uint8_t sin_channel, uint8_t sin_family, uint8_t type,
                           uint8_t protocol, uint32_t *sockfd, uint32_t *res);

void eth_channel_term (uint8_t data_channel);

void vec_stream_64_from_channel_term (Vec_Stream_64 *V0, uint8_t data_channel);

void vec_stream_64_to_channel_term (Vec_Stream_64 *V1, uint8_t data_channel);
```

A.18 Data Movement Functions

These functions support movement for data from the CPU processor, internal VLM memory, and Ethernet network.

A.18.1 DMA Functions for the CPU

```
void buffered_dma_cpu (int dir, int dma_path, void *obmaddr, int64_t obmstride, void *cmaddr,
                           int64_t cmstride, int64_t nbytes);

void streamed_dma_cpu_64 (Stream_64 *S, int dir, void *cmaddr, int64_t nbytes);
```

A.18.2 DMA Functions for the VLM

```
void streamed_dma_vlm_256 (Stream_256 *S, int dir, int vlm_port, int64_t address, int64_t
                           nbytes);
```

A.18.3 DMA Functions for the Ethernet Network

```
void vec_stream_64_from_eth_term (Vec_Stream_64 *V_in, int num_vecs);

void vec_stream_64_to_eth_term   (Vec_Stream_64 *V_out);
```

A.19 Pass Buffers

```
void pass_buffer_8_16          (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_32          (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_64          (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_128         (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_256         (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_512         (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_1024        (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_2048        (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_4096        (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_8_8192        (Stream_8 *S_in, Stream_8 *S_out, int64_t nbytes);
void pass_buffer_16_16         (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_32         (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_64         (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_128        (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_256        (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_512        (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_1024       (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_2048       (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_4096       (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_16_8192       (Stream_16 *S_in, Stream_16 *S_out, int64_t nbytes);
void pass_buffer_32_16         (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_32         (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_64         (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_128        (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_256        (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_512        (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_1024       (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
```

```
void pass_buffer_32_2048      (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_4096      (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_32_8192      (Stream_32 *S_in, Stream_32 *S_out, int64_t nbytes);
void pass_buffer_64_16        (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_32        (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_64        (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_128       (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_256       (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_512       (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_1024      (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_2048      (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_4096      (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_64_8192      (Stream_64 *S_in, Stream_64 *S_out, int64_t nbytes);
void pass_buffer_128_16       (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_32       (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_64       (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_128      (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_256      (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_512      (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_1024     (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_2048     (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_4096     (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_128_8192     (Stream_128 *S_in, Stream_128 *S_out, int64_t nbytes);
void pass_buffer_256_16       (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);
void pass_buffer_256_32       (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);
void pass_buffer_256_64       (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);
void pass_buffer_256_128      (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);
void pass_buffer_256_256      (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);
```

```
void pass_buffer_256_512       (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);

void pass_buffer_256_1024      (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);

void pass_buffer_256_2048      (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);

void pass_buffer_256_4096      (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);

void pass_buffer_256_8192      (Stream_256 *S_in, Stream_256 *S_out, int64_t nbytes);

void pass_buffer_128_16_term   (Stream_128 *S_in, Stream_128*S_out);

void pass_buffer_128_32_term   (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_64_term   (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_128_term  (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_256_term  (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_512_term  (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_1024_term (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_2048_term (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_4096_term (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_128_8192_term (Stream_128 *S_in, Stream_128 *S_out);

void pass_buffer_256_16_term   (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_32_term   (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_64_term   (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_128_term  (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_256_term  (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_512_term  (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_1024_term (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_2048_term (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_4096_term (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_256_8192_term (Stream_256 *S_in, Stream_256 *S_out);

void pass_buffer_8_16_term     (Stream_8 *S_in, Stream_8 *S_out);

void pass_buffer_8_32_term     (Stream_8 *S_in, Stream_8 *S_out);

void pass_buffer_8_64_term     (Stream_8 *S_in, Stream_8 *S_out);
```

```
void pass_buffer_8_128_term    (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_8_256_term    (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_8_512_term    (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_8_1024_term   (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_8_2048_term   (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_8_4096_term   (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_8_8192_term   (Stream_8 *S_in, Stream_8 *S_out);
void pass_buffer_16_16_term    (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_32_term    (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_64_term    (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_128_term   (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_256_term   (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_512_term   (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_1024_term  (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_2048_term  (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_4096_term  (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_16_8192_term  (Stream_16 *S_in, Stream_16 *S_out);
void pass_buffer_32_16_term    (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_32_term    (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_64_term    (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_128_term   (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_256_term   (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_512_term   (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_1024_term  (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_2048_term  (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_4096_term  (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_32_8192_term  (Stream_32 *S_in, Stream_32 *S_out);
void pass_buffer_64_16_term    (Stream_64 *S_in, Stream_64 *S_out);
```

```
void pass_buffer_64_32_term    (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_64_term    (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_128_term   (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_256_term   (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_512_term   (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_1024_term  (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_2048_term  (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_4096_term  (Stream_64 *S_in, Stream_64 *S_out);

void pass_buffer_64_8192_term  (Stream_64 *S_in, Stream_64 *S_out);
```

A.20 Large Selectors

There are two families of selector macros that may be useful in lieu of using conditional *if-else* constructs: key-based selectors and prioritized selectors.

A.20.1 Key-Based Selector Macros

The key-based selector macros implement a selector equivalent to a special case of a C switch statement.

```
switch (key & 3) {

   case 0 : vout = v0; break;

   case 1 : vout = v1; break;

   case 2 : vout = v2; break;

   case 3 : vout = v3; break;

   }
```

```
select_8bit_4val    (int key, int8_t v0, int8_t v1, ... int8_t v3, int8_t *vout);

select_8bit_8val    (int key, int8_t v0, int8_t v1, ... int8_t v7, int8_t *vout);

select_8bit_16val   (int key, int8_t v0, int8_t v1, ..., int8_t v15, int8_t *vout);

select_8bit_32val   (int key, int8_t v0, int8_t v1, ... int8_t v31, int8_t *vout);

select_8bit_64val   (int key, int8_t v0, int8_t v1, ... int8_t v63, int8_t *vout);

select_8bit_128val  (int key, int8_t v0, int8_t v1, ..., int8_t v127, int8_t *vout);

select_8bit_128val  (int key, int8_t v0, int8_t v1, ..., int8_t v255, int8_t *vout);

select_16bit_4val   (int key, int16_t v0, int16_t v1, ... int16_t v3, int16_t *vout);

select_16bit_8val   (int key, int16_t v0, int16_t v1, ... int16_t v7, int16_t *vout);
```

```
select_16bit_16val  (int key, int16_t v0, int16_t v1, ... int16_t v15, int16_t *vout);

select_16bit_32val  (int key, int16_t v0, int16_t v1, ... int16_t v31, int16_t *vout);

select_16bit_64val  (int key, int16_t v0, int16_t v1, ... int16_t v63, int16_t *vout);

select_16bit_128val (int key, int16_t v0, int16_t v1, ... int16_t v127, int16_t *vout);

select_16bit_256val (int key, int16_t v0, int16_t v1, ... int16_t v256, int16_t *vout);

select_32bit_4val   (int key, int v0, int v1, ... int v3, int *vout);

select_32bit_8val   (int key, int v0, int v1, ..., int v7, int *vout);

select_32bit_16val  (int key, int v0, int v1, ..., int v15, int *vout);

select_32bit_32val  (int key, int v0, int v1, ..., int v31, int *vout);

select_32bit_64val  (int key, int v0, int v1, ..., int v63, int *vout);

select_32bit_128val (int key, int v0, int v1, ..., int v127, int *vout);

select_64bit_4val   (int key, int64_t v0, int64_t v1, ..., int64_t v3, int *vout);

select_64bit_8val   (int key, int64_t v0, int64_t v1, ..., int64_t v7, int *vout);

select_64bit_16val  (int key, int64_t v0, int64_t v1, ..., int64_t v15, int *vout);

select_64bit_32val  (int key, int64_t v0, int64_t v1, ..., int64_t v31, int *vout);

select_64bit_64val  (int key, int64_t v0, int64_t v1, ..., int64_t v64, int *vout);

select_64bit_128val (int key, int64_t v0, int64_t v1, ..., int64_t v127, int *vout);

select_64bit_256val (int key, int64_t v0, int64_t v1, ..., int64_t v255, int *vout);
```

A.20.2 Prioritized Selector Macros

The prioritized selector macros implement a selector equivalent to an *if-else-if* sequence. These macros come in 8-, 16-, 32-, and 64-bit widths with 4, 8, 16, 32, 64, 128, and 256 selected values.

```
select_pri_8bit_4val    (int b0, int8_t v0, int b1, int8_t v1, int b2, int8_t v2, int8_t
                         def_val, int8_t *vout);

select_pri_8bit_8val    (int b0, int8_t v0, int b1, int8_t v1, . . ., int b6, int8_t v6,
                         int8_t def_val, int8_t *vout);

select_pri_8bit_16val   (int b0, int8_t v0, int b1, int8_t v1, . . ., int b14, int8_t v14,
                         int8_t def_val, int8_t *vout);

select_pri_8bit_32val   (int b0, int8_t v0, int b1, int8_t v1, . . ., int b30, int8_t v30,
                         int8_t def_val, int8_t *vout);

select_pri_8bit_64val   (int b0, int8_t v0, int b1, int8_t v1, . . ., int b62, int8_t v62,
                         int8_t def_val, int8_t *vout);
```

```
select_pri_8bit_128val  (int b0, int8_t v0, int b1, int8_t v1, . . ., int b126, int8_t v126,
                         int8_t def_val, int8_t *vout);

select_pri_8bit_256val  (int b0, int8_t v0, int b1, int8_t v1, . . ., int b254, int8_t v254,
                         int8_t def_val, int8_t *vout);

select_pri_16bit_4val   (int b0, int16_t v0, int b1, int16_t v1, int b2, int16_t v2, int16_t
                         def_val, int16_t *vout);

select_pri_16bit_8val   (int b0, int16_t v0, int b1, int16_t v1, ..., int b6, int16_t v6,
                         int16_t def_val, int16_t *vout);

select_pri_16bit_16val  (int b0, int16_t v0, int b1, int16_t v1, ..., int b14, int16_t v14,
                         int16_t def_val, int16_t *vout);

select_pri_16bit_32val  (int b0, int16_t v0, int b1, int16_t v1, ..., int b30, int16_t v30,
                         int16_t def_val, int16_t *vout);

select_pri_16bit_64val  (int b0, int16_t v0, int b1, int16_t v1, ..., int b62, int16_t v62,
                         int16_t def_val, int16_t *vout);

select_pri_16bit_128val (int b0, int16_t v0, int b1, int16_t v1, ..., int b126, int16_t v126,
                         int16_t def_val, int16_t *vout);

select_pri_16bit_256val (int b0, int16_t v0, int b1, int16_t v1, ..., int b254, int16_t v254,
                         int16_t def_val, int16_t *vout);

select_pri_32bit_4val   (int b0, int v0, int b1, int v1, ..., int b2, int v2, int def_val, int
                         *vout);

select_pri_32bit_8val   (int b0, int v0, int b1, int v1, ..., int b6, int v6, int def_val, int
                         *vout);

select_pri_32bit_16val  (int b0, int v0, int b1, int v1, ..., int b14, int v14, int def_val,
                         int *vout);

select_pri_32bit_32val  (int b0, int v0, int b1, int v1, ..., int b30, int v30, int def_val,
                         int *vout);

select_pri_32bit_64val  (int b0, int v0, int b1, int v1, ..., int b62, int v62, int def_val,
                         int *vout);

select_pri_32bit_128val (int b0, int v0, int b1, int v1, ..., int b126, int v126, int def_val,
                         int *vout);

select_pri_32bit_256val (int b0, int v0, int b1, int v1, ..., int b254, int v254, int def_val,
                         int *vout);

select_pri_64bit_4val   (int b0, int64_t v0, int b1, int64_t v1, int b2, int64_t v2, int64_t
                         def_val, int64_t *vout);

select_pri_64bit_8val   (int b0, int64_t v0, int b1, int64_t v1, ... , int b6, int64_t v6,
                         int64_t def_val, int64_t *vout);

select_pri_64bit_16val  (int b0, int64_t v0, int b1, int64_t v1, ..., int b14, int64_t v14,
                         int64_t def_val, int64_t *vout);
```

```
select_pri_64bit_32val  (int b0, int64_t v0, int b1, int64_t v1, ... , int b30, int64_t v30,
                         int64_t def_val, int64_t *vout);

select_pri_64bit_64val  (int b0, int64_t v0, int b1, int64_t v1, ... , int b62, int64_t v62,
                         int64_t def_val, int64_t *vout);

select_pri_64bit_128val (int b0, int64_t v0, int b1, int64_t v1, ... , int b126, int64_t v126,
                         int64_t def_val, int64_t *vout);

select_pri_64bit_256val (int b0, int64_t v0, int b1, int64_t v1, ... , int b254, int64_t v254,
                         int64_t def_val, int64_t *vout);
```

A.21 Debug/Simulation Trace Settings

A.21.1 *ulogic sim MAPTRACEMASK* Values

```
DEBUG_MAP_BASIC          (1)
DEBUG_SCA_TRACE          (2)
DEBUG_DUMP_TRACE_ON_EXIT (4)
DEBUG_SNAPSIM            (8)
DEBUG_SSIM_PAYLOAD       (0x10)
DEBUG_SSIM_MEM           (0x20)
DEBUG_SSIM_REGS          (0x40)
DEBUG_USIM               (0x80)
DEBUG_USIM_DC            (0x100)
DEBUG_USIM_DMA           (0x200)
DEBUG_USIM_OBM           (0x400)
DEBUG_USIM_DATA          (0x800)
DEBUG_USIM_STREAM        (0x1000)
DEBUG_USIM_BIG_DMA       (0x2000)
DEBUG_USIM_REQ           (0x4000)
DEBUG_USIM_ETHER         (0x8000)
DEBUG_USIM_VLM           (0x10000)
DEBUG_BARRIERS           (0x20000)
```

A.21.2 *debug mode MAPTRACEMASK* Values

```
DEBUG_MAP_BASIC                  (1)
DEBUG_BRIDGE                     (2)
DEBUG_BARRIERS                   (4)
DEBUG_STREAM_DATA                (8)
DEBUG_STREAM_CALL                (0x10)
DEBUG_BUFFERED_CPU_IN            (0x20)
DEBUG_BUFFERED_CPU_OUT           (0x40)
DEBUG_BUFFERED_GCM_IN            (0x80)
DEBUG_BUFFERED_GCM_OUT           (0x100)
DEBUG_STREAM_DATA_IN_FROM_GCM    (0x200)
DEBUG_STREAM_DATA_INTO_OBM       (0x400)
DEBUG_STREAM_DATA_OUT_TO_GCM     (0x800)
DEBUG_STREAM_DATA_OUT_OF_OBM     (0x1000)
DEBUG_STREAM_COMPLEX             (0x2000)
DEBUG_STREAM_ETHER               (0x4000)
```

APPENDIX B DIRECTSTREAM FLOATING POINT OPERATORS

DirectStream has developed a set of floating point macros with both single precision and double precision accuracy. The *add*, *subtract*, *multiply*, *divide*, and *square root* macros utilize VHDL cores supplied by Altera.

These floating point macros closely implement the IEEE 754 standard, except for handling of special number inputs such as Denormals, NaNs, and Infinity. When these special numbers are input, the macro may or may not return a compliant result. The Altera VHDL cores implement a round to the nearest even mode and the maximum error is ±1ulp.

Table 27 contains the supported floating point macros:

Table 27 Supported Floating Point Macros

	Single	Double
Operation	**Version 1**	**Version 1**
Add	Y	Y
Subtract	Y	Y
Multiply	Y	Y
Divide	Y	Y
Modulus	Y	Y
Absolute	Y	Y
Square Root	Y	Y
Square	Y	Y
Negate	Y	Y
Minimum	Y	Y
Maximum	Y	Y
Multiply Accumulate	Y	Y
Accumulate	Y	Y
Equal To	Y	Y
Greater Than	Y	Y
Greater Than or Equal To	Y	Y
Less Than	Y	Y
Less Than or Equal To	Y	Y
Not Equal	Y	Y

B.1 Rounding

Intel uses round-to-nearest rounding mode as the default mode. The Intel processor has two different floating point execution environments: x87 Floating Point Unit (FPU) and Single Instruction, Multiple Data (SIMD) computation instructions. The Carte compiler does not internally generate the SIMD instructions (Streaming SIMD Extensions [SSE] and SSE2), which uses the round-to-zero (truncate) mode for some instructions.

B.2 Denormal Numbers

Denormal numbers are fractional numbers that represent the set of very small numbers on either side of zero. They are represented by floating point numbers with zero as the exponent and a non-zero in the significand.

B.3 Intel Sign Invert

The *f32 > i32* or *f32 > i64* test cases truncate very large positive floating point numbers. This is basically an overflow condition. Executing on the Intel processor changes the sign in these cases, whereas executing on the MAP processor keeps the sign the same. There is not an IEEE requirement for how the overflow is handled. DirectStream believes that keeping the same sign is the correct method for this conversion.

B.4 Relational Operations

There are no known differences between the Intel and MAP implementation of the relational operations. The following operations are supported: ==, *!*=, >, >=, <, and <=.

APPENDIX D MAKEFILE TEMPLATE AND EXAMPLES

D.1 Makefile Template

```
# ------------------------------------------------------------------
# User source files , and executable name
# ------------------------------------------------------------------
FILES           = <list of micro-processor routines>
MAPFILES        = <list of MAP routines>
BIN             = <executable name>

# ------------------------------------------------------------------
# Multi chip parameters
# (Leave commented out if not used)
# ------------------------------------------------------------------
#PRIMARY        = <primary file 1>   <primary file 2>
#SECONDARY      = <secondary file 1> <secondary file 2>
#CHIP2          = <file to compile to user chip 2>

# ------------------------------------------------------------------
# User supplied directory of MAP routines to inline
# ------------------------------------------------------------------
#INLINEDIR      =

# ------------------------------------------------------------------
# User supplied macros
#
# (Leave commented out if not used)
# (Old style. MAP_H only)
# ------------------------------------------------------------------
#MACROS         = <directory-name/macro-file>
#MY_BLKBOX      = <directory-name/blackbox-file>
#MY_NGO_DIR     = <directory-name>
#MY_INFO        = <directory-name/info-file>
#
# (New style. MAP_J and up)
# (Macro, Debug and Info files must have same basenames)
# ------------------------------------------------------------------
#USER_MACROS    = <directory-name/macro-file.v>

# ------------------------------------------------------------------
# Definitions and Flags for compilers and linkers
# ------------------------------------------------------------------
CC              = gcc
LD              = gcc

MCCFLAGS        = -v
#CPPFLAGS       = -I my_inc     # C Pre-Processor flags
#CFLAGS         = -g            # C Compiler flags
#LDFLAGS        =               # Loader flags
#LOADLIBES      = -L lib_path   # Extra library search path
#LDLIBS         = -l my_lib     # Extra library name

# ------------------------------------------------------------------
# VCS simulation settings
# (Set as needed, otherwise just leave commented out)
# ------------------------------------------------------------------
#VCSDUMP        = yes   # YES or yes to generate vcd+ trace dump
```

```
# ------------------------------------------------------------------------
# No modifications are required below
# ------------------------------------------------------------------------
MAKIN   ?= $(MC_ROOT)/opt/srcci/comp/lib/AppRules.make
include $(MAKIN)
```

D.2 Makefile Tailoring Examples

D.2.1 C Application

```
# ------------------------------------------------------------------------
# User source files, and executable name
# ------------------------------------------------------------------------
FILES           = main.c
MAPFILES        = poly.mc
BIN             = poly

# ------------------------------------------------------------------------
# Multi chip parameters
# (Leave commented out if not used)
# ------------------------------------------------------------------------
#PRIMARY        = <primary file 1>   <primary file 2>
#SECONDARY      = <secondary file 1> <secondary file 2>
#CHIP2          = <file to compile to user chip 2>

# ------------------------------------------------------------------------
# User supplied directory of MAP routines to inline
# ------------------------------------------------------------------------
#INLINEDIR      =

# ------------------------------------------------------------------------
# User supplied macros
#
# (Leave commented out if not used)
# (Old style. MAP_H only)
# ------------------------------------------------------------------------
#MACROS         = <directory-name/macro-file>
#MY_BLKBOX      = <directory-name/blackbox-file>
#MY_NGO_DIR     = <directory-name>
#MY_INFO        = <directory-name/info-file>
#
# (Macro, Debug and Info files must have same basenames)
# ------------------------------------------------------------------------
#USER_MACROS    = <directory-name/macro-file.v>

# ------------------------------------------------------------------------
# Definitions and Flags for compilers and linkers
# ------------------------------------------------------------------------
CC              = gcc
LD              = gcc

MCCFLAGS        = -v
#CPPFLAGS       = -I my_inc     # C Pre-Processor flags
#CFLAGS         = -g            # C Compiler flags
#LDFLAGS        =               # Loader flags
#LOADLIBES      = -L lib_path   # Extra library search path
#LDLIBS         = -l my_lib     # Extra library name

# ------------------------------------------------------------------------
```

```
# VCS simulation settings
# (Set as needed, otherwise just leave commented out)
# ------------------------------------------------------------------
#VCSDUMP        = yes   # YES or yes to generate vcd+ trace dump

# ------------------------------------------------------------------
# No modifications are required below
# ------------------------------------------------------------------
MAKIN   ?= $(MC_ROOT)/opt/srcci/comp/lib/AppRules.make
include $(MAKIN)
```

D.2.2 C Application with User-Defined Macros

```
# ----------------------------------
# User defines FILES, MAPFILES, and BIN here
# ----------------------------------
FILES           = main.c
MAPFILES        = poly.c
BIN             = poly

# ----------------------------------
# User defined macros info supplied here
# (Leave commented out if not used)
# ----------------------------------
USER MACROS     = my_macro/myop.v

# ----------------------------------
# User supplied MCC flags
# ----------------------------------
MCCFLAGS        = -v
# ----------------------------------
# User supplied flags for C compilers
# ----------------------------------
CC              = gcc
LD              = gcc
CFLAGS          =
# ----------------------------------
# VCS simulation settings
# (Set as needed, otherwise leave commented out)
# ----------------------------------

#VCSDUMP        = yes # YES or yes to generate vcd+ trace dump
# ----------------------------------
# No modifications are required below
# ----------------------------------
MAKIN          ?= $(MC_ROOT)/opt/srcci/comp/lib/AppRules.make
include $(MAKIN)
D.2.3  C Applicatio
```

D.2.3 Multiple MAP Routines and Multiple User-Defined Macros

```
# ------------------------------------------------------------------
# User source files, and executable name
# ------------------------------------------------------------------
FILES           = main.c \
                  func.c
MAPFILES        = poly_a.mc \
                  poly_b.mc
```

```
BIN             = poly

# ------------------------------------------------------------------
# Multi chip parameters
# (Leave commented out if not used)
# ------------------------------------------------------------------
#PRIMARY        = <primary file 1>   <primary file 2>
#SECONDARY      = <secondary file 1> <secondary file 2>

# ------------------------------------------------------------------
# User supplied directory of MAP routines to inline
# ------------------------------------------------------------------
#INLINEDIR      =

# ------------------------------------------------------------------
# User supplied macros
#
# (Macro, Debug and Info files must have same basenames)
# ------------------------------------------------------------------
USER_MACROS     = my_macro/my_op_a.v \
                  my_macro/my_op_b.v

# ------------------------------------------------------------------
# Definitions and Flags for compilers and linkers
# ------------------------------------------------------------------
CC              = gcc
LD              = gcc

MCCFLAGS        = -v
CPPFLAGS        = -DMORE_COWBELL # C Pre-Processor flags
#CFLAGS         = -g             # C Compiler flags
#LDFLAGS        =                # Loader flags
#LOADLIBES      = -L lib_path    # Extra library search path
#LDLIBS         = -l my_lib      # Extra library name

# ------------------------------------------------------------------
# VCS simulation settings
# (Set as needed, otherwise just leave commented out)
# ------------------------------------------------------------------
#VCSDUMP        = yes   # YES or yes to generate vcd+ trace dump

# ------------------------------------------------------------------
# No modifications are required below
# ------------------------------------------------------------------
MAKIN   ?= $(MC_ROOT)/opt/srcci/comp/lib/AppRules.make
include $(MAKIN)
```

APPENDIX E GLOSSARY

Table 28 contains definitions of commonly used terms and acronyms in this document.

Table 28 Acronym Definitions

Acronym	Definition
CC	Control Chip
CM	Chassis Manager
CM2OBM	CPU to On-Board Memory
CPU	Central Processing Unit
DDR	Double Data Rate
DDR2	Double Data Rate 2
DDR3	Double Data Rate 3
DMA	Direct Memory Access
ECC	Error Correction Code
FPGA	Field-Programmable Gate Array
FPU	Floating Point Unit
FIFO	First In First Out
GB	Gigabyte
GbE	Gigabit Ethernet
HDD	Hard Disk Drive
HDL	Hardware Description Language
HTTP	HyperText Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
LUT	Look Up Table
MAC	Media Access Control
MB	Megabytes
MHz	Megahertz
NFS	Network File System
OBM	On-Board Memory
OBM2CM	On-Board Memory to CPU
OpenMP	Open Multi-Processing
OS	Operating System
RAM	Random Access Memory
SCM	System Common Memory
SDRAM	Synchronous Dynamic Random Access Memory
SECDED	Single Error Correction, Double Error Detection
SGMII	Serial Gigabit Media Independent Interface
SIMD	Single Instruction, Multiple Data
SODIMM	Small Outline Dual In-Line Memory Module
SRAM	Static Random Access Memory
SSD	Solid State Drive
SSE	Streaming SIMD Extensions
TBD	To Be Determined
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UL	User Logic
VHDL	VHSIC Hardware Description Language

Acronym	Definition
VHSIC	Very High Speed Integrated Circuit
VLM	Very-Large Memory
VPD	Virtual Private Database