Application Acceleration with High-Level Synthesis

Lab A No.11_Streaming_free_running_k2k

110061585 Tim Hsu

Streaming_free_running_k2k

- The free-running kernel has no memory input or output port, and therefore it interacts with the host or other kernels (other kernels can be regular kernel or another free-running kernel) only through streams.
- When the FPGA is programmed by the binary container (xclbin), the free-running kernel starts running on the FPGA, and therefore it does not need to be started from the host code.
- The kernel works on the stream data as soon as it starts receiving from the platform I/O or other kernels, and it stalls when the data is not available.

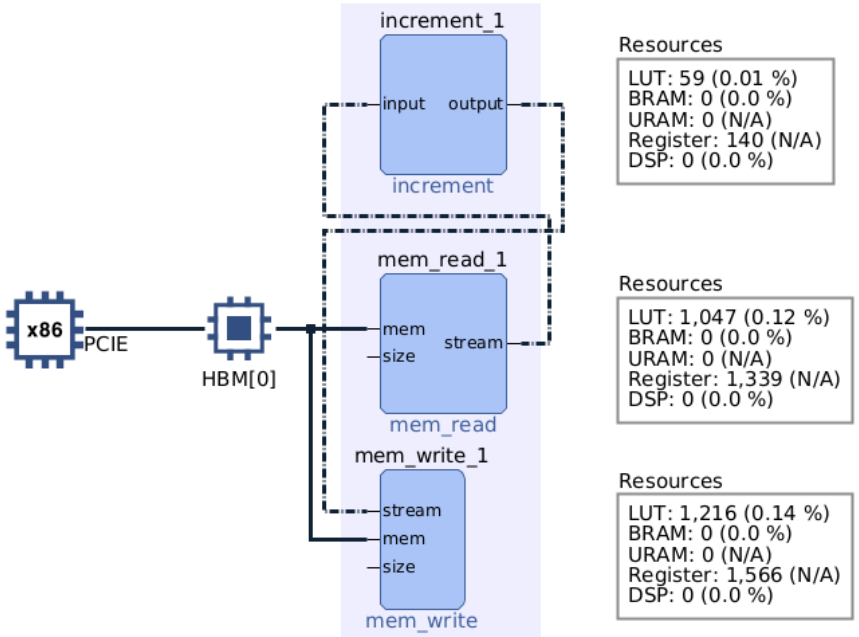
Kernel to kernel streaming example consisting of three compute units in a linear hardware pipeline.

1) Memory read
   This Kernel reads the input vector from Global Memory and streams its output.
2) Increment
   This Kernel reads stream input, increments the value and streams to output.
3) Memory write
   This Kernel reads from its input stream and writes into Global Memory

For free running kernel, user needs to specify ap_ctrl_none for return port.
This will create the kernel without AXI lite interface. Kernel will always be in running states

```
extern "C" {
void increment(hls::stream<ap_axiu<32, 0, 0, 0> >& input, hls::stream<ap_axiu<32, 0, 0, 0> >& output) {
#pragma HLS interface ap_ctrl_none port = return

increment:
    while (1) {
        ap_axiu<32, 0, 0, 0> v = input.read();
        v.data = v.data + 1;
        output.write(v);
        if (v.last) {
            break;
        }
    }
}
}
```

Example of system diagram.



Example of resource utilization.

Kernel Route Utilization

Name	LUT	LUTAsMem	REG	BRAM	URAM	DSP
Platform	101076	8631	125165	178	0	4
User Budget	768940	393385	1618195	1166	640	5936
Used Resources	2322	627	3045	0	0	0
Unused Resources	766618	392758	1615150	1166	640	5936
increment (1)	59	0	140	0	0	0
increment_1	59	0	140	0	0	0
mem_read (1)	1047	282	1339	0	0	0
mem_read_1	1047	282	1339	0	0	0
mem_write (1)	1216	345	1566	0	0	0
mem_write_1	1216	345	1566	0	0	0

Streaming_k2k_mm

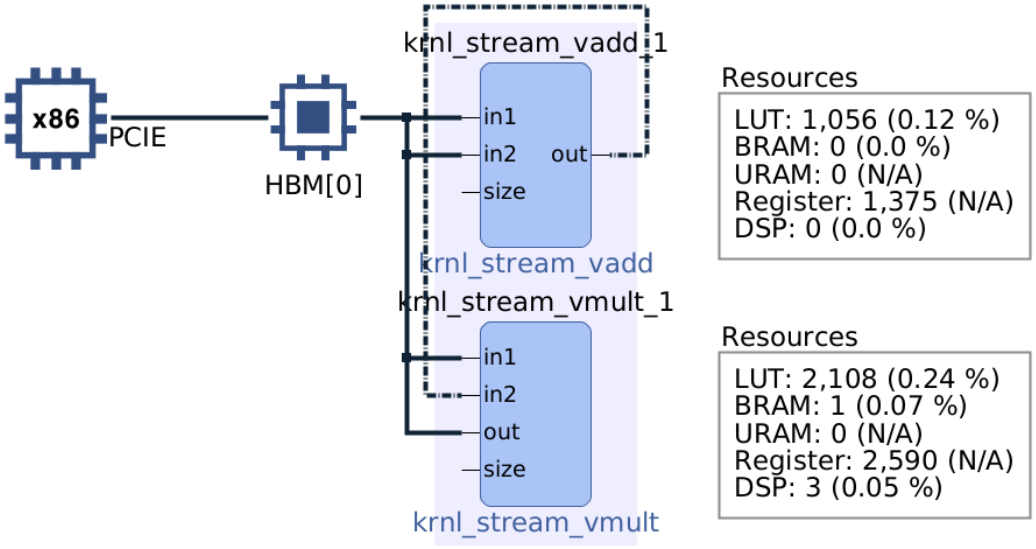
the free-running kernel only contains `hls::stream` inputs and outputs. The recommended coding guidelines include:

- Use `hls::stream<ap_axiu<D,0,0,0> >` if the port is interacting with another stream port from the kernel.
- Use `hls::stream<qdma_axis<D,0,0,0> >` if the port is interacting with the host.
- Use the `hls::stream` data type for the function parameter causes Vitis HLS to infer an AXI4-Stream port (`axis`) for the interface.
- The free-running kernel must also specify the following special INTERFACE pragma

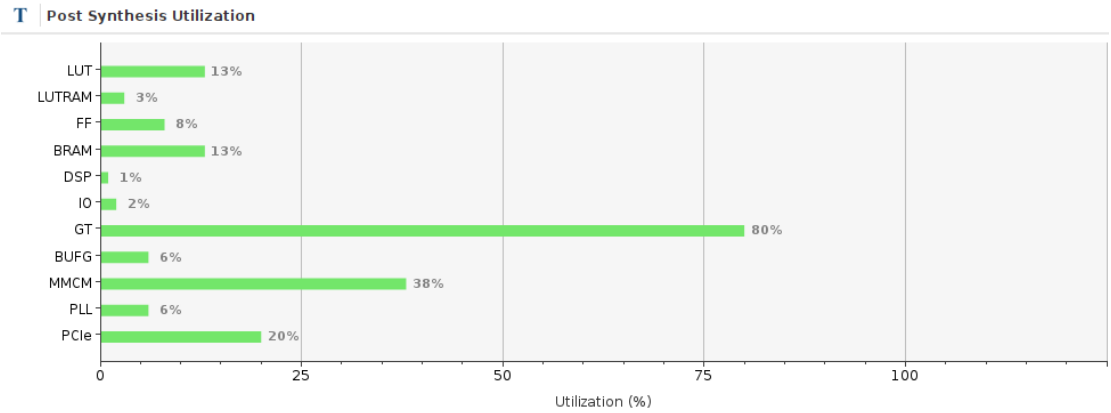
On the code below it reset the data vectors and run the kernel after it allocate buffer in global memory. Moreover, Buffers are allocated CL_MEM_USE_HOST_PTR for efficient memory and Device-to-host communication. When the Kernel Arguments have been set, it also copy input data to device global memory for launching the Kernel after that it copy Result from Device Global Memory to Host Local Memory then open CL Host Code Ends to compare the device results with software results.

```
reset(h_a.data(), h_b.data(), h_c.data(), sw_results.data(), hw_results.data(), size);

unsigned int vector_size_bytes = size * sizeof(int);
OCL_CHECK(err, cl::Buffer buffer_in1(context, CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY, vector_size_bytes, h_a.data(),
                                     &err));
OCL_CHECK(err, cl::Buffer buffer_in2(context, CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY, vector_size_bytes, h_b.data(),
                                     &err));
OCL_CHECK(err, cl::Buffer buffer_in3(context, CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY, vector_size_bytes, h_c.data(),
                                     &err));
OCL_CHECK(err, cl::Buffer buffer_output(context, CL_MEM_USE_HOST_PTR | CL_MEM_WRITE_ONLY, vector_size_bytes,
                                        hw_results.data(), &err));

OCL_CHECK(err, err = krnl_vadd.setArg(0, buffer_in1));
OCL_CHECK(err, err = krnl_vadd.setArg(1, buffer_in2));
OCL_CHECK(err, err = krnl_vadd.setArg(3, size));

OCL_CHECK(err, err = krnl_vmult.setArg(0, buffer_in3));
OCL_CHECK(err, err = krnl_vmult.setArg(2, buffer_output));
OCL_CHECK(err, err = krnl_vmult.setArg(3, size));

OCL_CHECK(err, err = q.enqueueMigrateMemObjects({buffer_in1, buffer_in2, buffer_in3}, 0));
OCL_CHECK(err, err = q.enqueueTask(krnl_vadd));
OCL_CHECK(err, err = q.enqueueTask(krnl_vmult));
q.finish();
OCL_CHECK(err, err = q.enqueueMigrateMemObjects({buffer_output}, CL_MIGRATE_MEM_OBJECT_HOST));
q.finish();
bool match = verify(sw_results.data(), hw_results.data(), size);

return (match ? EXIT_SUCCESS : EXIT_FAILURE);
```

Example of system diagram.



Example of resource utilization.



Resource	Utilization	Available	Utilization %
LUT	116542	870016	13.40
LUTRAM	11257	402016	2.80
FF	131557	1743360	7.55
BRAM	179.50	1344	13.36
DSP	7	5940	0.12
IO	10	416	2.40
GT	16	20	80.00
BUFG	39	672	5.80
MMCM	3	8	37.50
PLL	1	16	6.25
PCIe	1	5	20.00