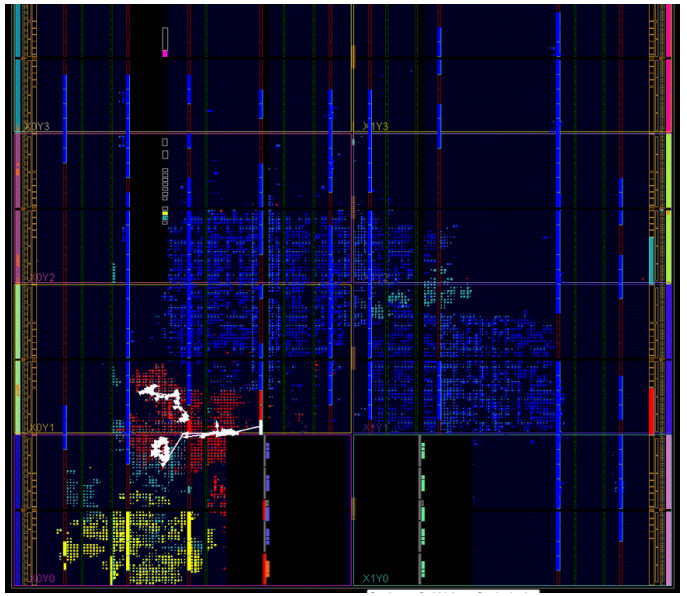
Firmware Implementation and SDAccel

Dylan Rankin (MIT)



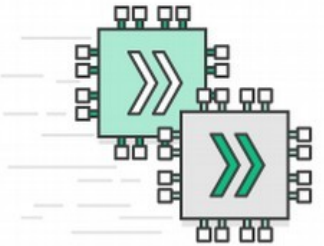
Fast Machine Learning Workshop

September 12th, 2019

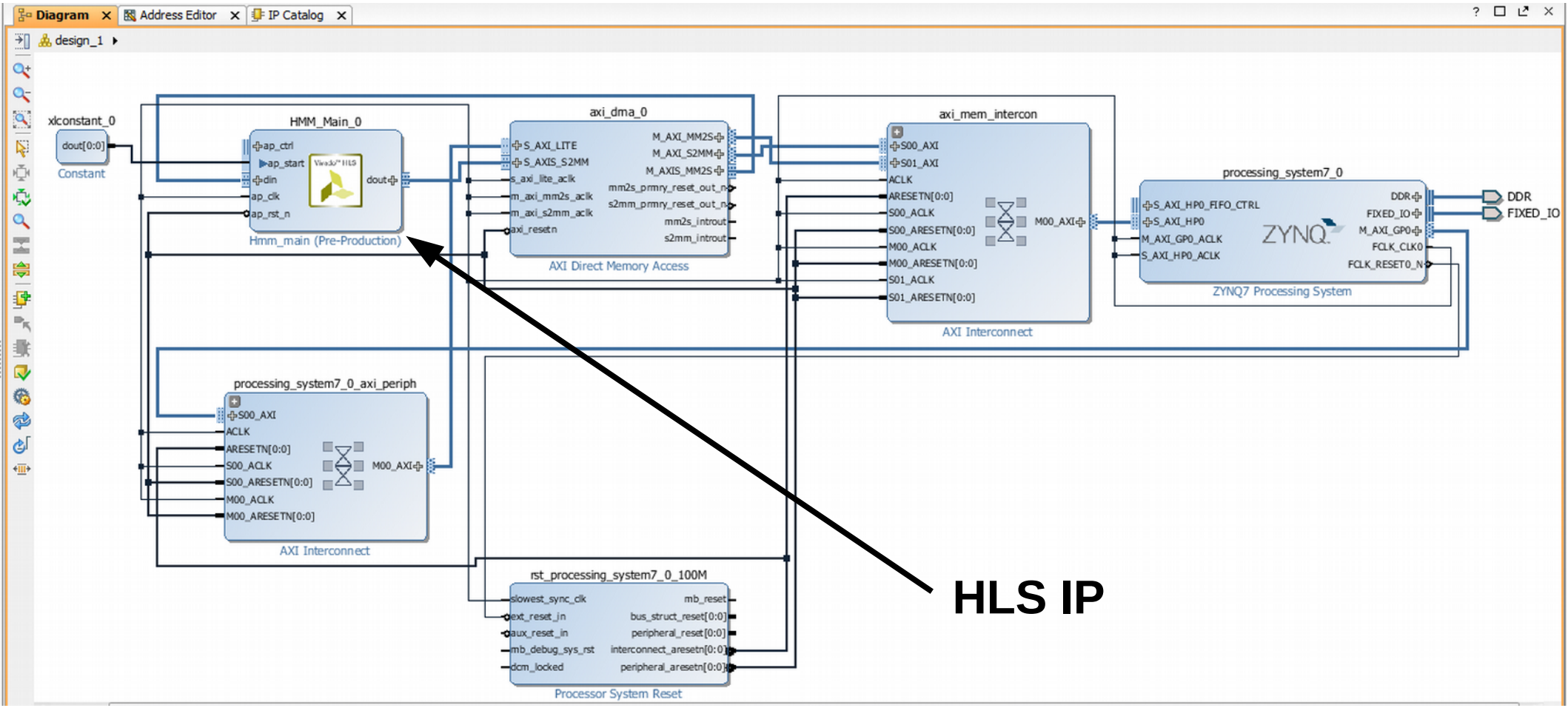
FPGA

Introduction

- HLS project is just one part of actually running an application on an FPGA
  - Need to handle data in to/out of FPGA, how to actually route signals through FPGA, etc.
- SDAccel is a tool that helps in development/implementing designs on FPGAs specifically for acceleration (CPU ↔ FPGA)
- Will show how to use SDAccel to accelerate an hls4ml project

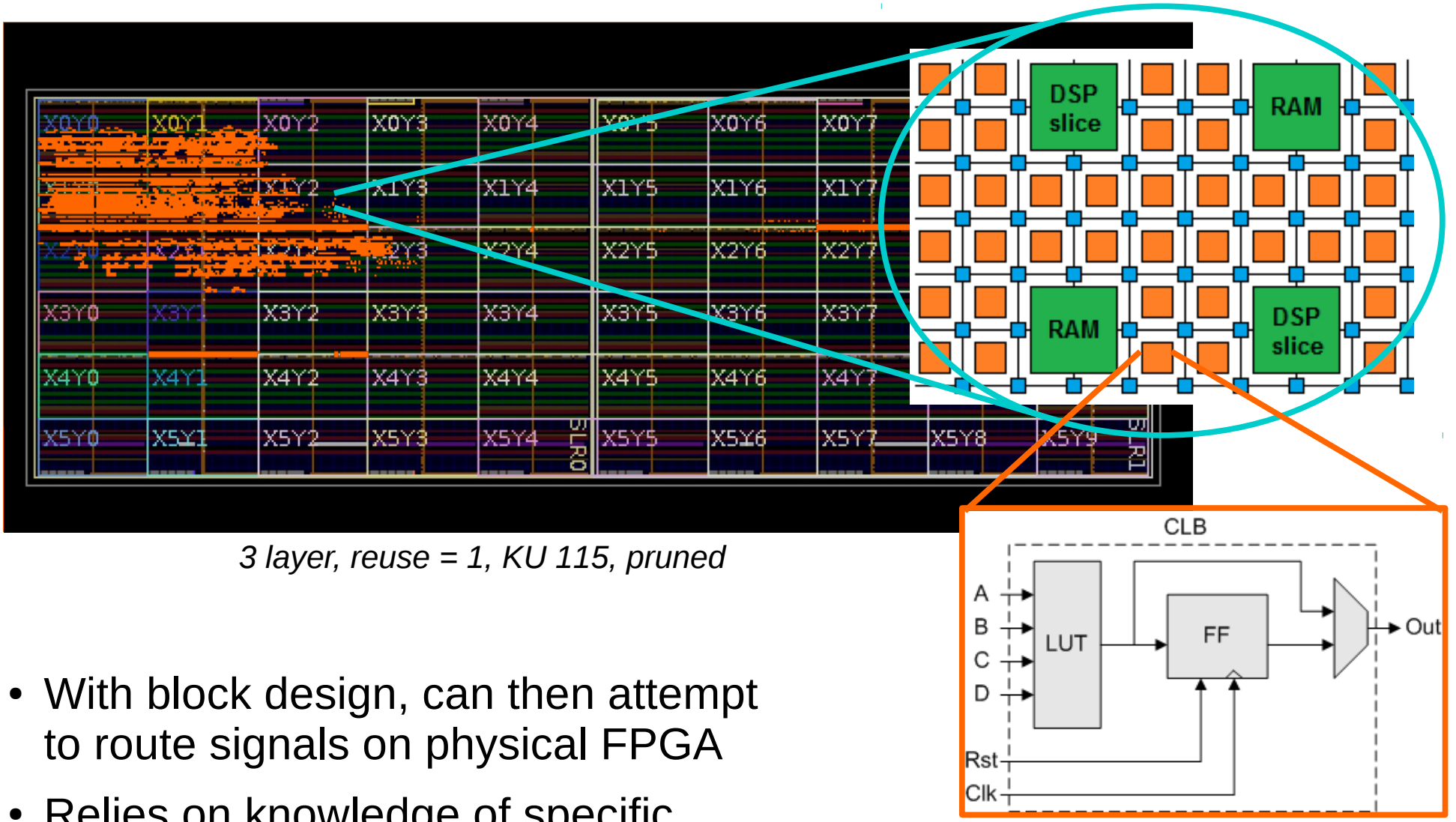


Programming an FPGA



- Typically need to specify how signals will be sent through FPGA

Programming an FPGA



3 layer, reuse = 1, KU 115, pruned

- With block design, can then attempt to route signals on physical FPGA
- Relies on knowledge of specific device, layout of components

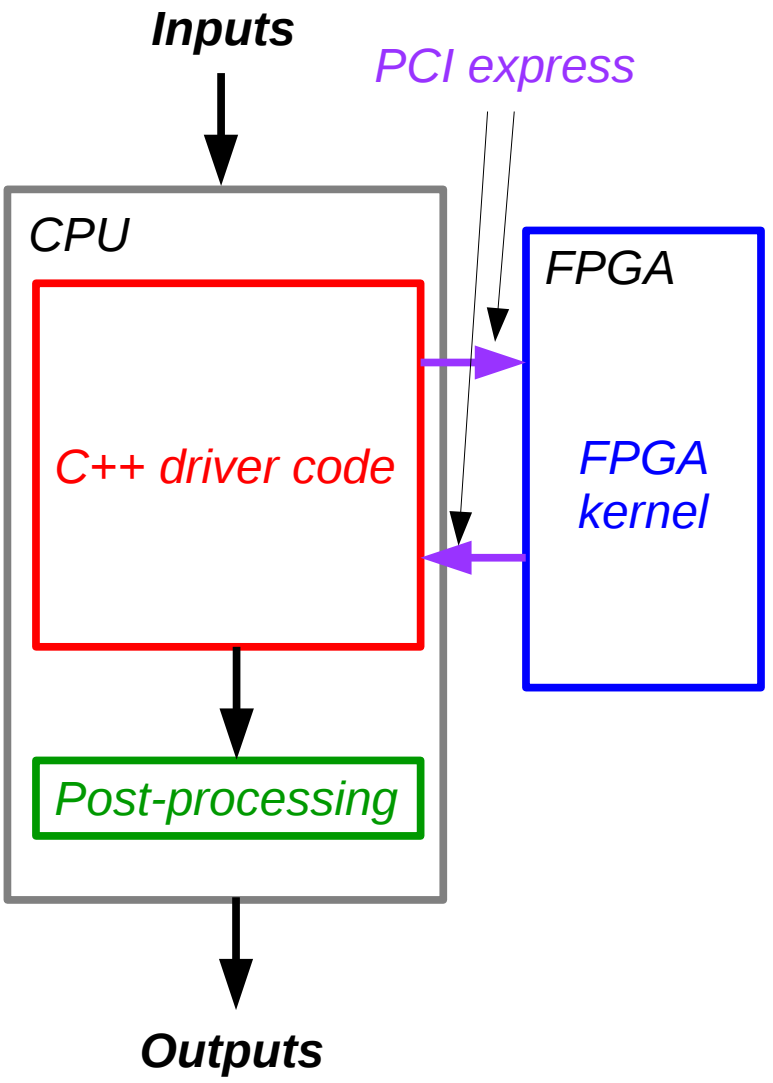
aws Overview

- AWS F1 instances are machines connected directly to a Xilinx Virtex UltraScale+ FPGA (VU9P) using PCI-express
- General application development on AWS done using SDAccel

Virtex Ultrascale+ VU9P
6800 DSPs
1M LUTs
2M FFs
75 Mb BRAM

SDAccel™ Environment

- SDAccel Development Environment allows the development/running of connected FPGA kernels and CPU processes
- Define FPGA kernel using HLS, OpenCL, or VHDL/Verilog
  - Need to meet certain design constraints regarding control, input/output protocol
- Any FPGA application defined in one of these languages can be easily accelerated using SDAccel
- Once FPGA kernel is available, write host code to run on CPU and manage data transfer, FPGA execution
  - Examples: https://github.com/Xilinx/SDAccel_Examples



SDAccel Makefile

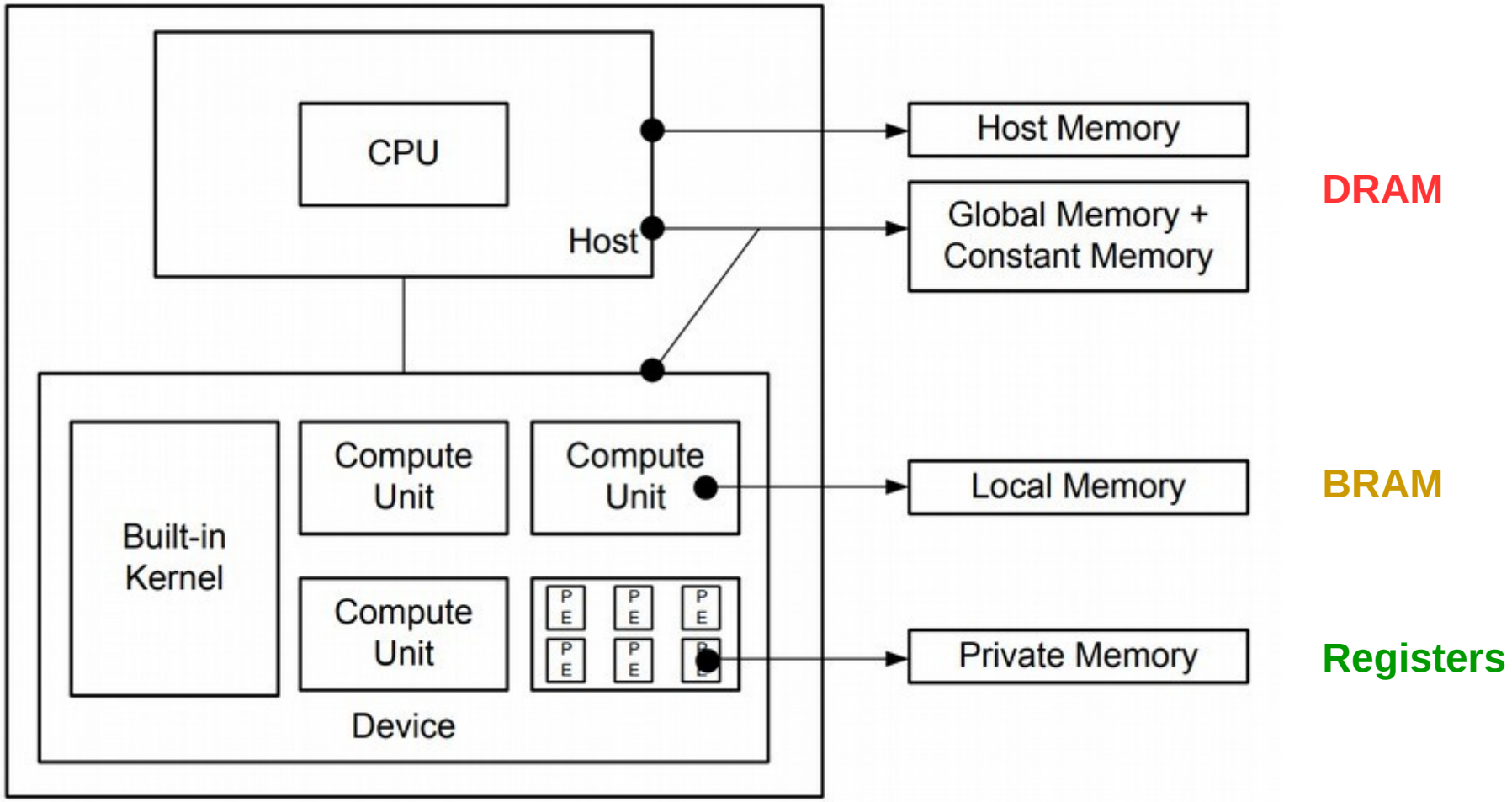
- SDAccel uses make commands to build executables
  - xcpp for the CPU host code (Xilinx C++ compiler)
  - xocc for the FPGA kernel (Xilinx OpenCL Compiler)
    - xocc will run Vivado HLS under the hood
- Three main running modes (all compile host code):
- Software Emulation : **`sw_emu`**
  - Emulate kernel in software (checks for C errors, ~csim)
- Hardware Emulation : **`hw_emu`**
  - Emulate kernel in hardware (builds kernel, ~cosim)
- Hardware : **`hw`**
  - Create xclbin (system image file, contains bitstream with kernel for programming FPGA, ~implementation/routing/bitstream)

FPGA Kernel

- SDAccel kernels require inputs and outputs to be passed in certain manner
  - Must be AXI-stream
  - Must be mapped to global memory
  - Specific control

```
extern "C" {
void aws_hls4ml(
        const data_t *in, // Read-Only Vector
        data_t *out       // Output Result
        )
{
#pragma HLS INTERFACE m_axi port=in  offset=slave bundle=gmem
#pragma HLS INTERFACE m_axi port=out offset=slave bundle=gmem
#pragma HLS INTERFACE s_axilite port=in   bundle=control
#pragma HLS INTERFACE s_axilite port=out  bundle=control
#pragma HLS INTERFACE s_axilite port=return bundle=control
```

SDAccel Data Management



- Transfer between CPU and FPGA uses DRAM

FPGA Kernel

```
    unsigned short insize, outsize; //necessary for hls4ml kernel, not used

    input_t in_buf[STREAMSIZE][N_INPUTS];
    result_t out_buf[STREAMSIZE][N_OUTPUTS]; //these will get partitioned properly in
the hls4ml code

//getting data from axi stream and formatting properly
    for (int i = 0; i < STREAMSIZE; i++) {
#pragma HLS LOOP UNROLL
        for (int j = 0; j < N_INPUTS; j++) {
#pragma HLS LOOP UNROLL
            in_buf[i][j] = (input_t)in[i*N_INPUTS+j];
        }
    }

//run inference
    for (int i = 0; i < STREAMSIZE; i++) {
#pragma HLS dataflow
        Hls4ml: myproject(in_buf[i],out_buf[i],insize,outsize);
    }

//place output into axi stream output
    for (int i = 0; i < STREAMSIZE; i++) {
#pragma HLS LOOP UNROLL
        for (int j = 0; j < N_OUTPUTS; j++) {
#pragma HLS LOOP UNROLL
            out[i*N_OUTPUTS+j] = (data_t)out_buf[i][j];
        }
    }
```

Host Code

- Need to allocate block in memory for data
- Pass information to device with OpenCL buffer objects
  - `cl::Buffer`

```
size_t vector_size_in_bytes = sizeof(data_t) * DATA_SIZE_IN * STREAMSIZE;
size_t vector_size_out_bytes = sizeof(data_t) * DATA_SIZE_OUT * STREAMSIZE;
std::vector<data_t,aligned_allocator<data_t>> source_in(DATA_SIZE_IN*STREAMSIZE);
std::vector<data_t,aligned_allocator<data_t>> source_hw_results(DATA_SIZE_OUT*STREAMSIZE);

// OPENCL HOST CODE AREA START
// get_xil_devices() is a utility API which will find the xilinx
// platforms and will return list of devices connected to Xilinx platform
std::vector<cl::Device> devices = xcl::get_xil_devices();
cl::Device device = devices[0];
cl::Context context(device);
cl::CommandQueue q(context, device, CL_QUEUE_PROFILING_ENABLE);

// Allocate Buffer in Global Memory
// Buffers are allocated using CL_MEM_USE_HOST_PTR for efficient memory and
// Device-to-host communication
cl::Buffer buffer_in   (context,CL_MEM_USE_HOST_PTR | CL_MEM_READ_ONLY,
        vector_size_in_bytes, source_in.data());
cl::Buffer buffer_output(context,CL_MEM_USE_HOST_PTR | CL_MEM_WRITE_ONLY,
        vector_size_out_bytes, source_hw_results.data());
```

Host Code

- Specify binary file to use for programming FPGA
- Connect global memory buffers with kernel arguments

```
// find_binary_file() is a utility API which will search the xclbin file for
// targeted mode (sw_emu/hw_emu/hw) and for targeted platforms.
std::string binaryFile = xcl::find_binary_file(device_name,"aws_hls4ml");
// import_binary_file() ia a utility API which will load the binaryFile
// and will return Binaries.
cl::Program::Binaries bins = xcl::import_binary_file(binaryFile);
devices.resize(1);
cl::Program program(context, devices, bins);
std::vector<cl::Memory> inBufVec, outBufVec;
inBufVec.push_back(buffer_in);
outBufVec.push_back(buffer_output);
cl::Kernel krnl_aws_hls4ml(program,"aws_hls4ml");
int narg = 0;
krnl_aws_hls4ml.setArg(narg++, buffer_in);
krnl_aws_hls4ml.setArg(narg++, buffer_output);
```

Host Code

- To run:
- 1) Place inputs in allocated global memory
- 2) Launch kernel
- 3) Move outputs back from global memory when available

```
    // Copy input data to device global memory
    q.enqueueMigrateMemObjects(inBufVec,0/* 0 means from host*/);
    // Launch the Kernel
    // For HLS kernels global and local size is always (1,1,1). So, it
is recommended
    // to always use enqueueTask() for invoking HLS kernel
    q.enqueueTask(krnl_aws_hls4ml);
    // Copy Result from Device Global Memory to Host Local Memory
    q.enqueueMigrateMemObjects(outBufVec,CL_MIGRATE_MEM_OBJECT_HOST);
    // Check for any errors from the command queue
    q.finish();
```