HW#5 Domain-Specific Accelerator

Chun-Jen Tsai
NYCU
12/08/2023

Homework Goal

- In this homework, you must integrate a domain-specific accelerator (DSA) to Aquila to improve the speed of an MLP neural network

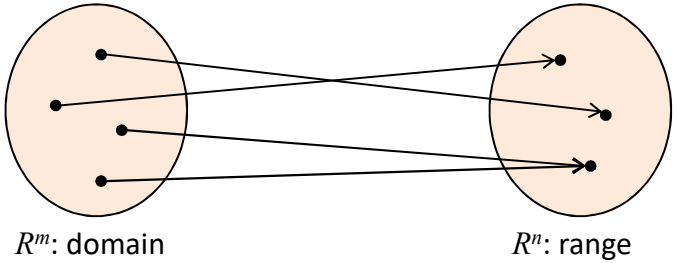
- Your tasks:
  - Add a vector floating-point HW IP by Xilinx into the Aquila SoC
  - Use the IP to accelerate the computing speed of a neural network application

- You should upload report & code to E3 by 1/5, 17:00.

Neural Network as Computers

- All computing systems are used to compute functions:

$$f: R^m \rightarrow R^n$$

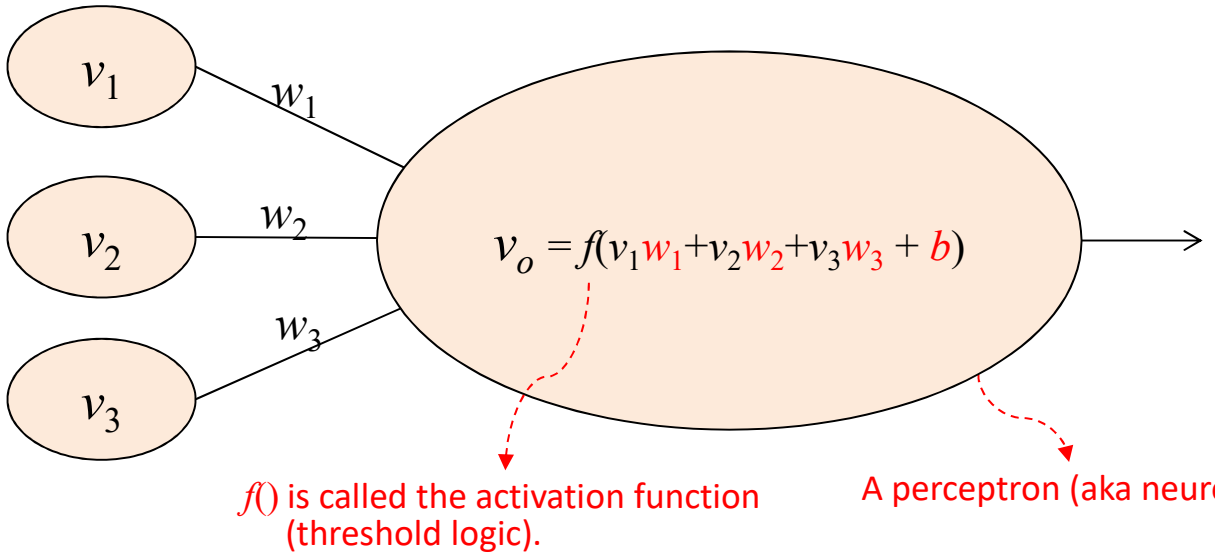


- In 1957, Kolmogorov proved that any continuous function can be decomposed into $(2m+1)+n$ different $R^m \rightarrow R$ linear functions

† A. N. Kolmogorov, "On the representation of continuous function of many variables by superpositions of continuous functions of one variable and addition," *Doklady Akademii Nauk USSR*, 114(5):953-956, 1957.

Basic Neural Network Components

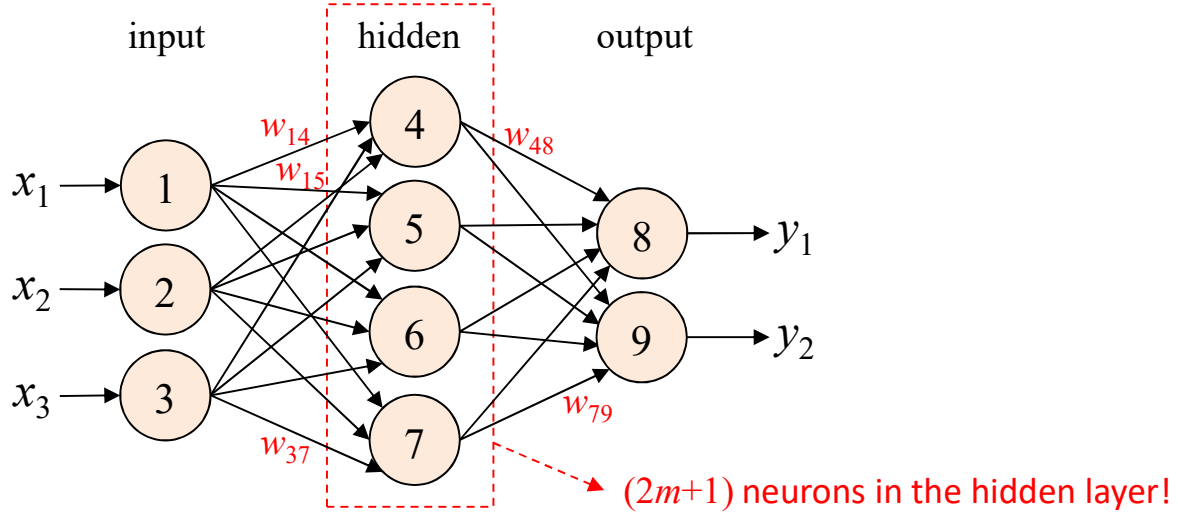
- W. McCulloch and W. Pitts proposed a computing element, threshold logic, for Artificial Neural Network (ANN) in 1943:



† W. McCulloch and W. Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. **5** (4), 1943, pp. 115–133.

A Universal Neural Network

- By A. N. Kolmogorov, a continuous function f: $R^m \rightarrow R^n$ can be computed using 3 layers of perceptrons



  - The "program" of a neural network is in the form of neural link weights, $\{ w_{ij} \}$

Hand-Written Character Recognition

- In this homework, we will use a multi-layer perceptrons (MLP) for hand-written character recognition
  - The MLP has 3 layers with 784, 48, 10 neurons in each layer
- The neural network weights (i.e. the program) is trained by the MNIST dataset:
  - Each image has 28×28 pixels

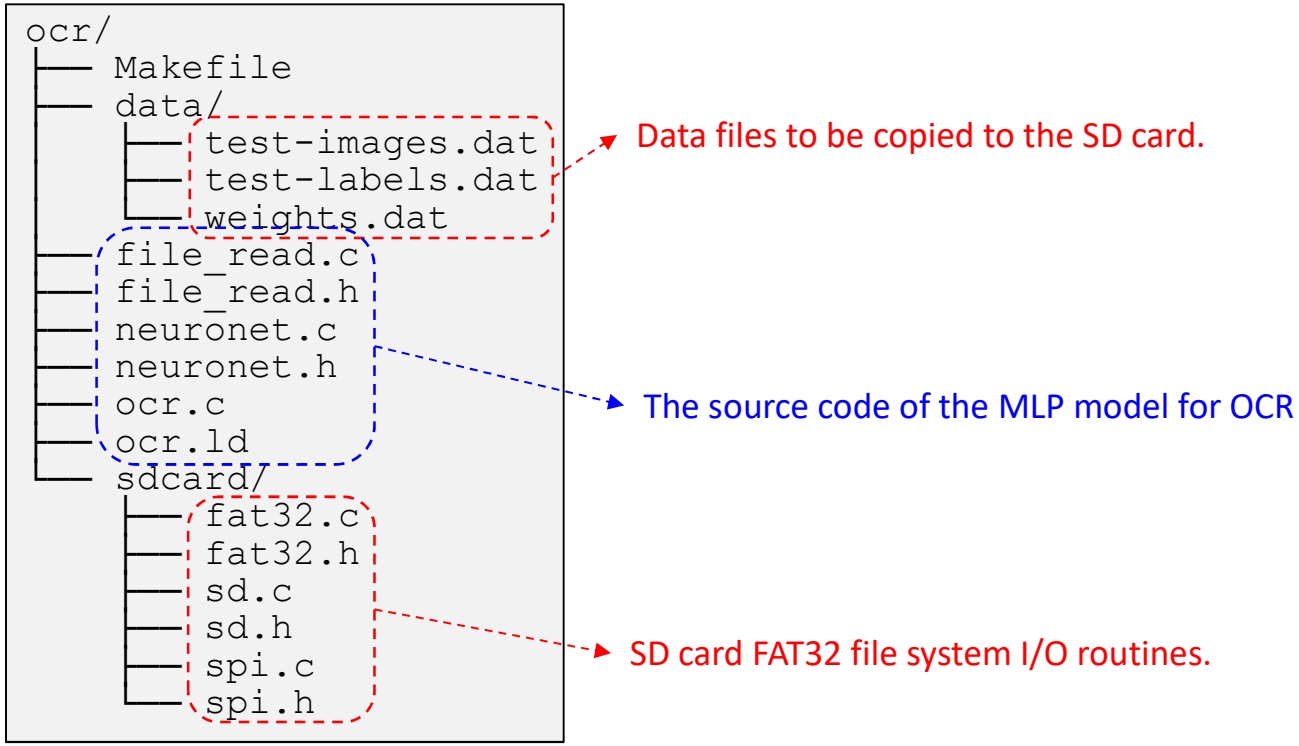
MLP Layer Design for MNIST Data

- Since the MLP has 1D input layer, we must convert 2D image input to 1D input:
  - Using the scanline order to do the conversion: $R^{28\times 28} \rightarrow R^{784\times 1}$
  - Therefore, we need 784 input neurons
- The output layer shows the “likelihood” of each digits
  - A reasonable choice is to use 10 output neurons
  - The maximal neuron gives us the most likely digit in the image
- The # of hidden layer neurons is a tough choice
  - A tradeoff between accuracy and complexity
  - Can be chosen by trial-and-err†

† To train a model with format that can be used in this HW, check out: https://github.com/AdamYuan/SimpleNN.

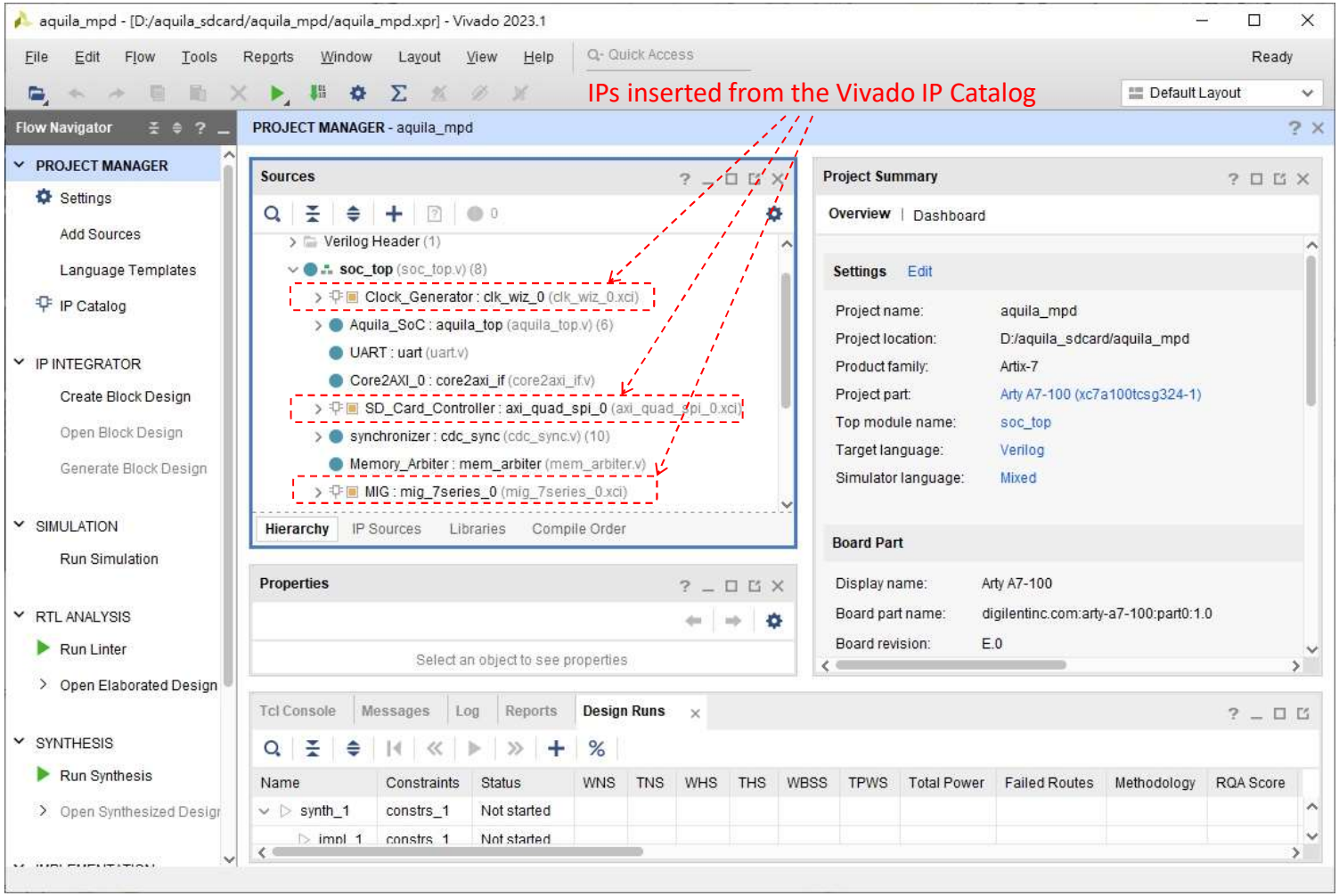
The HW/SW for HW#5

- You should download from E3 the HW workspace, aquila_sdcard.zip and the MLP program, ocr.tgz.
- The source tree of the ocr program:



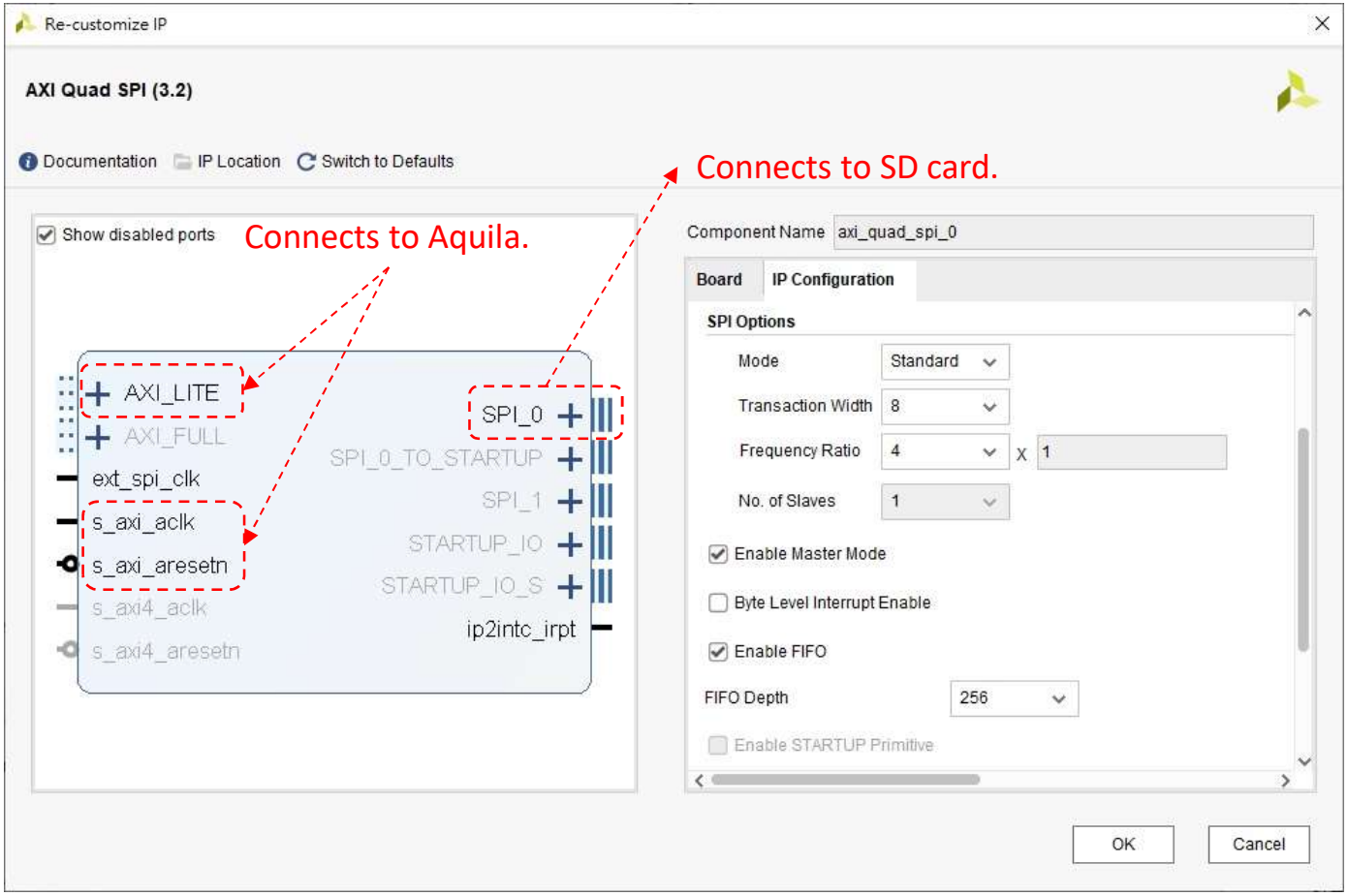
The HW Workspace for HW#5

- We have added an AXI SPIO controller for SD card:



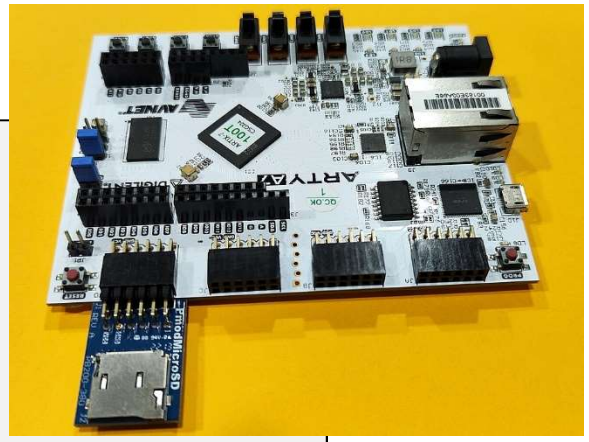
AXI Quad SPI Controller

- Double-click the IP opens the parameter box:



Instantiation of the IP in Aquila

- Instantiation of the SPI Controller:



```
// ---------------------------------
//  SPI controller
// ---------------------------------
//  This controller connects to the PMOD microSD module in
//  the JD connector of the Arty A7-100T.
//
axi_quad_spi_0 SD_Card_Controller(

    // Interface ports to the Aquila SoC.
    .s_axi_aclk(clk),
    .s_axi_aresetn(~rst),
    .s_axi_awaddr(axi_awaddr),
    .s_axi_awvalid(axi_awvalid),     // master signals write addr/ctrl valid.
    .s_axi_awready(axi_awready),     // slave ready to fetch write address.
    .s_axi_wdata(axi_wdata),         // write data to the slave.
    .s_axi_wstrb(axi_wstrb),         // byte select signal for write operation.
    .s_axi_wvalid(axi_wvalid),       // master signals write data is valid.
    .s_axi_wready(axi_wready),       // slave ready to accept the write data.
    .s_axi_araddr(axi_araddr),
    .s_axi_arready(axi_arready),     // slave ready to fetch read address.
    .s_axi_arvalid(axi_arvalid),     // master signals read addr/ctrl valid.
    .s_axi_bready(axi_bready),       // master is ready to accept the response.
    .s_axi_bresp(axi_bresp),         // reponse code from the slave.
    .s_axi_bvalid(axi_bvalid),       // slave has sent the respond signal.
    .s_axi_rdata(axi_rdata),         // read data from the slave.
    .s_axi_rready(axi_rready),       // master is ready to accept the read data.
    .s_axi_rresp(axi_rresp),         // slave sent read response.
    .s_axi_rvalid(axi_rvalid),       // slave signals read data ready.

    // Interface ports to the SD Card.
    .ext_spi_clk(clk),
    .io0_o(spi_mosi),
    .io1_i(spi_miso),
    .sck_o(spi_sck),
    .ss_o(spi_ss)
);
```

The Aquila Device Interface

- The Aquila core uses a simple memory-mapped I/O interface to talk to the external devices
  - A bridge is required for it to talk to the AXI SPI controller

```
aquila_top Aquila_SoC
(
    .clk_i(clk), .rst_i(rst), .base_addr_i(32'b0),

    // External instruction memory ports.
    .M_IMEM_strobe_o(IMEM_strobe),
              . . .
    .M_IMEM_data_i(IMEM_data),

    // External data memory ports.
    .M_DMEM_strobe_o(DMEM_strobe),
              . . .
    .M_DMEM_data_i(DMEM_rd_data),

    // I/O device ports.
    .M_DEVICE_strobe_o(dev_strobe),    // Issue read/write requests.
    .M_DEVICE_addr_o(dev_addr),        // Target device address.
    .M_DEVICE_rw_o(dev_we),            // Read or write?
    .M_DEVICE_byte_enable_o(dev_be),   // Byte-select signal.
    .M_DEVICE_data_o(dev_din),         // Data input to the device.
    .M_DEVICE_data_ready_i(dev_ready), // Is device ready?
    .M_DEVICE_data_i(dev_dout)         // Data output from the device.
);
```

Bridging the IP Interface

- Most IPs in the Xilinx IP Catalog use the AXI bus interfaces to communicate with other IPs:
  - AXI
    - Full bus: enable both burst and single-beat data transfer
    - Lite bus: enable single-beat data transfer
  - AXI Stream: enable burst-only data transfer

- We must convert the Aquila interface bus signals to the AXI bus signals for IP integration
  - The module `core2axi_if.v` is used for Aquila to connect to any IP that supports AXI Lite bus interface.

Running the OCR Program

1. First, copy the data files to SD card
2. Insert the SD card into the SD card daughter card
3. Insert the daughter card into the socket JD on Arty
4. Then, build the software `ocr.elf` by typing “make”
5. Synthesize the Aquila SoC and configure the FPGA
6. Load and run `ocr.elf`

Output of the OCR Program

- The OCR program does hand-written digits recognition:

```
================================================================================
Copyright (c) 2019-2023, EISL@NYCU, Hsinchu, Taiwan.
The Aquila SoC is ready.
Waiting for an ELF file to be sent from the UART ...

Program entry point at 0x80001EFC, size = 0x9414.
--------------------------------------------------------------------------------

(1) Reading the test images, labels, and neural weights.
It took 5255 msec to read files from the SD card.

(2) Perform the hand-written digits recognition test.
Here, we use a 3-layer 784-48-10 MLP neural network model.
Begin computing ... tested 100 images. The accuracy is 85.00%

It took 21904 msec to perform the test.

--------------------------------------------------------------------------------
Program exit with a status code 0
Press <reset> on the FPGA board to reboot the cpu ...
```

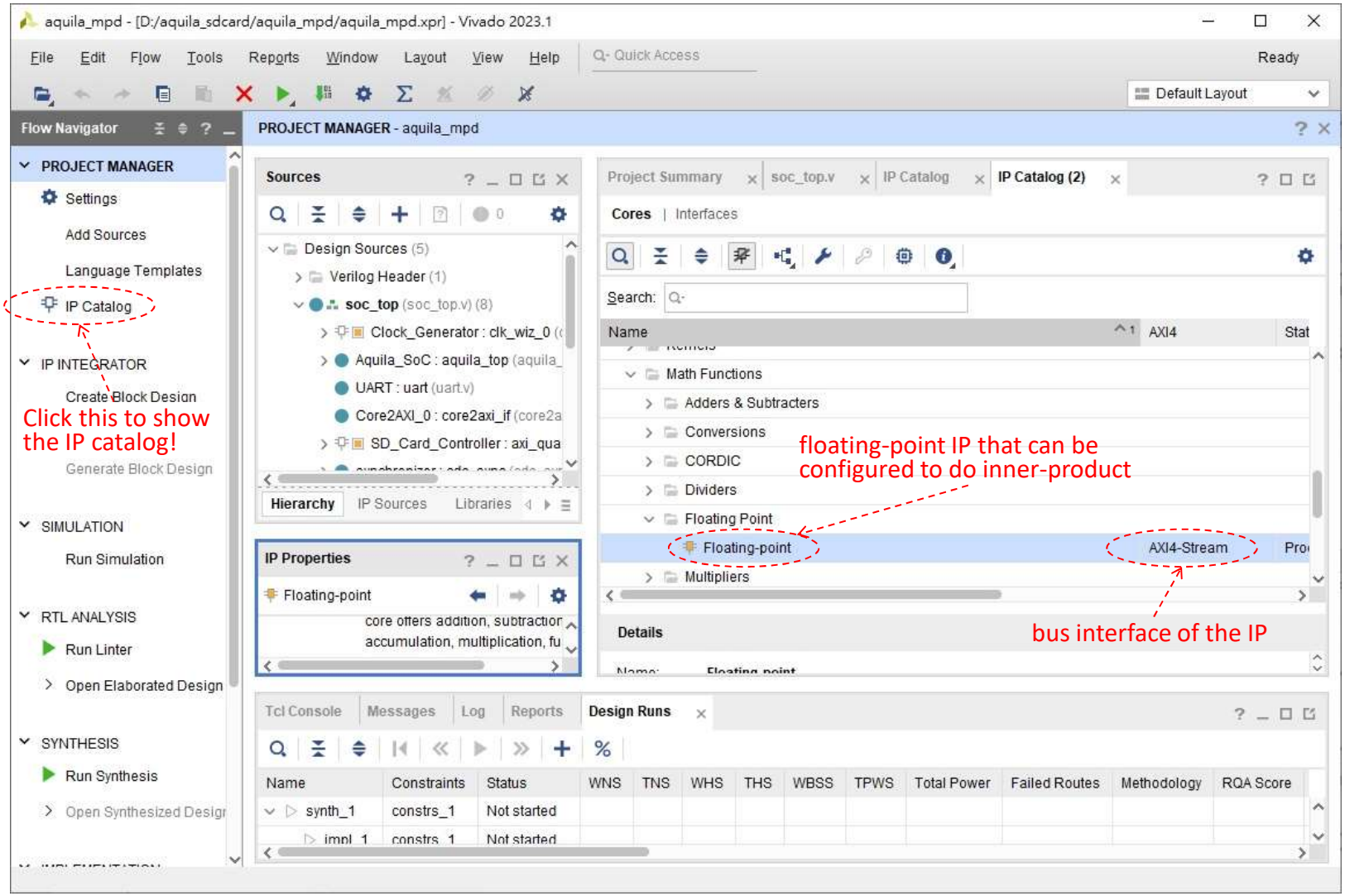
Key Hotspot of the Program

- In `neuronet.c`, the: function `neuronet_eval`:

```
int neuronet_eval(NeuroNet *nn, float *images)
{
        . . . . .

    // Forward computations
    neuron_idx = nn->n_neurons[0];
    for (layer_idx = 1; layer_idx < nn->total_layers; layer_idx++)
    {
        for (idx = 0; idx < nn->n_neurons[layer_idx]; idx++, neuron_idx++)
        {
            // 'p_weight' points to the first forward weight of a layer.
            p_weight = nn->forward_weights[neuron_idx];
            inner_product = 0.0;

            // Loop over all forward-connected neural links.
            p_neuron = nn->previous_neurons[neuron_idx];
            for (int jdx = 0; jdx < nn->n_neurons[layer_idx-1]; jdx++)
            {
                inner_product += (*p_neuron++) * (*p_weight++);
            }

            inner_product += *(p_weight); // The last weight is the bias.
            nn->neurons[neuron_idx] = relu(inner_product);
        }
    }

       . . . . .

    return max_idx;
}
```

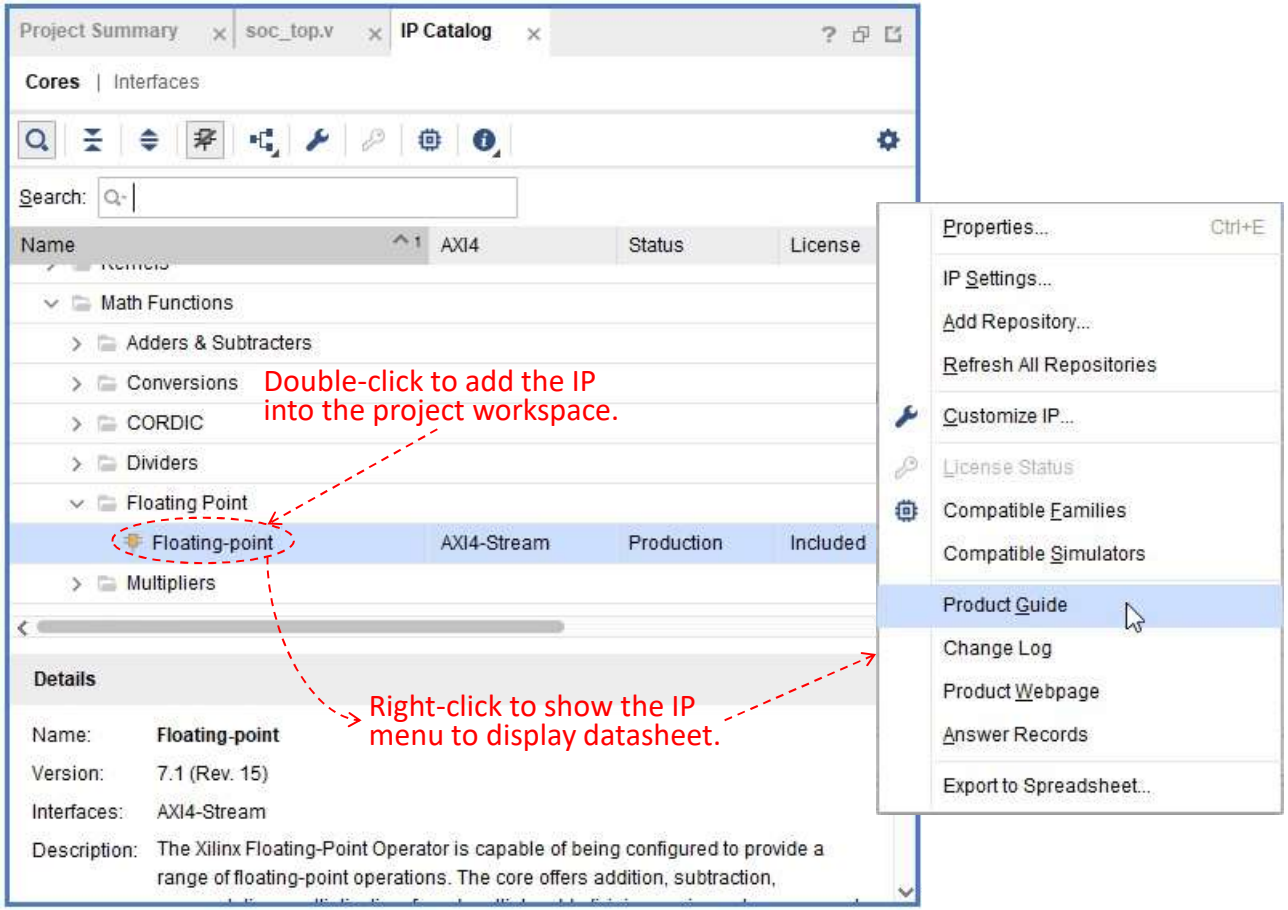
Inner product of two floating-point vectors!

Inner-Product IP in Xilinx IP Catalog



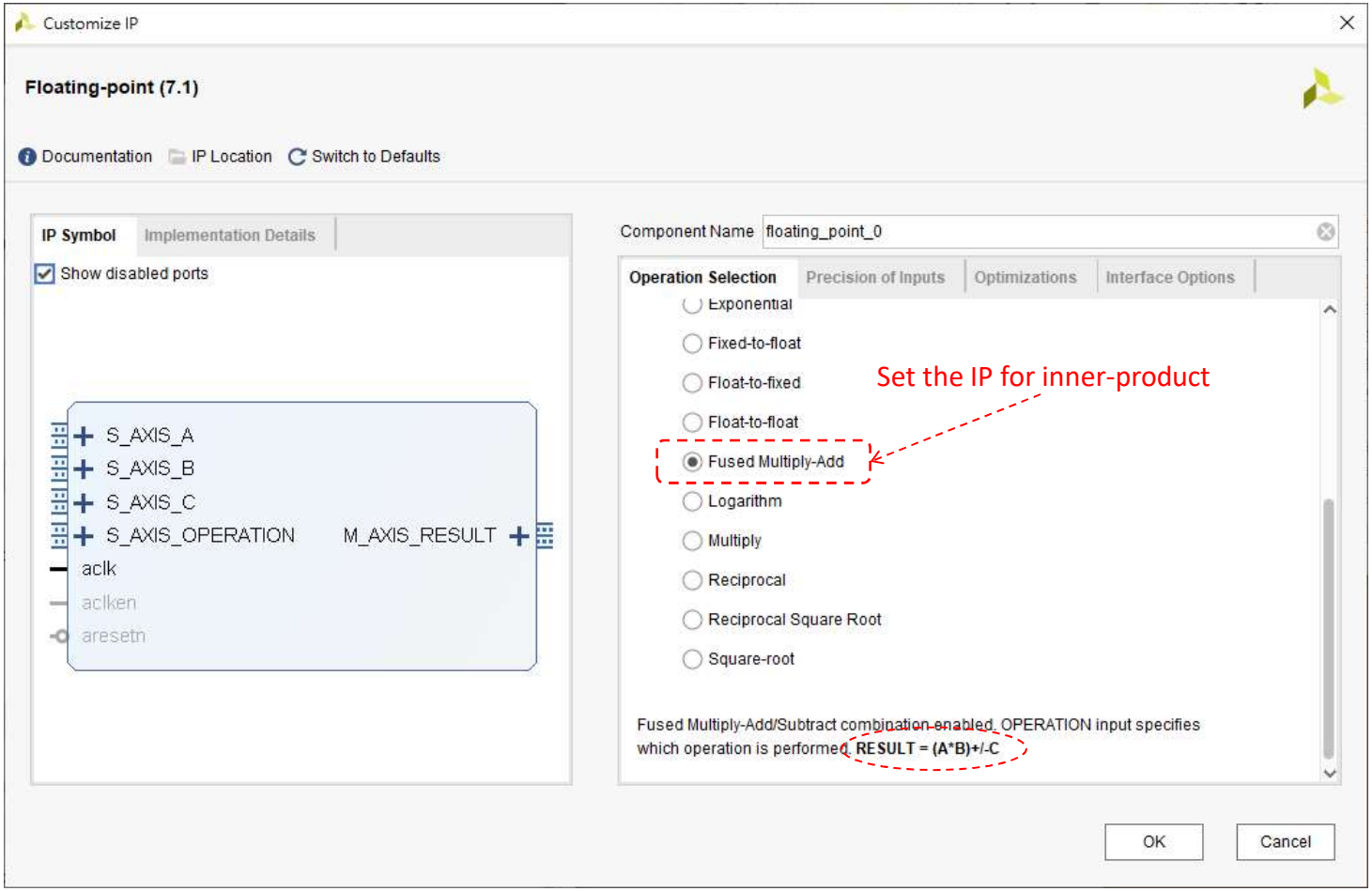
How to Add and IP into Aquila SoC

- Simply double-clicking the IP to add it to the workspace:



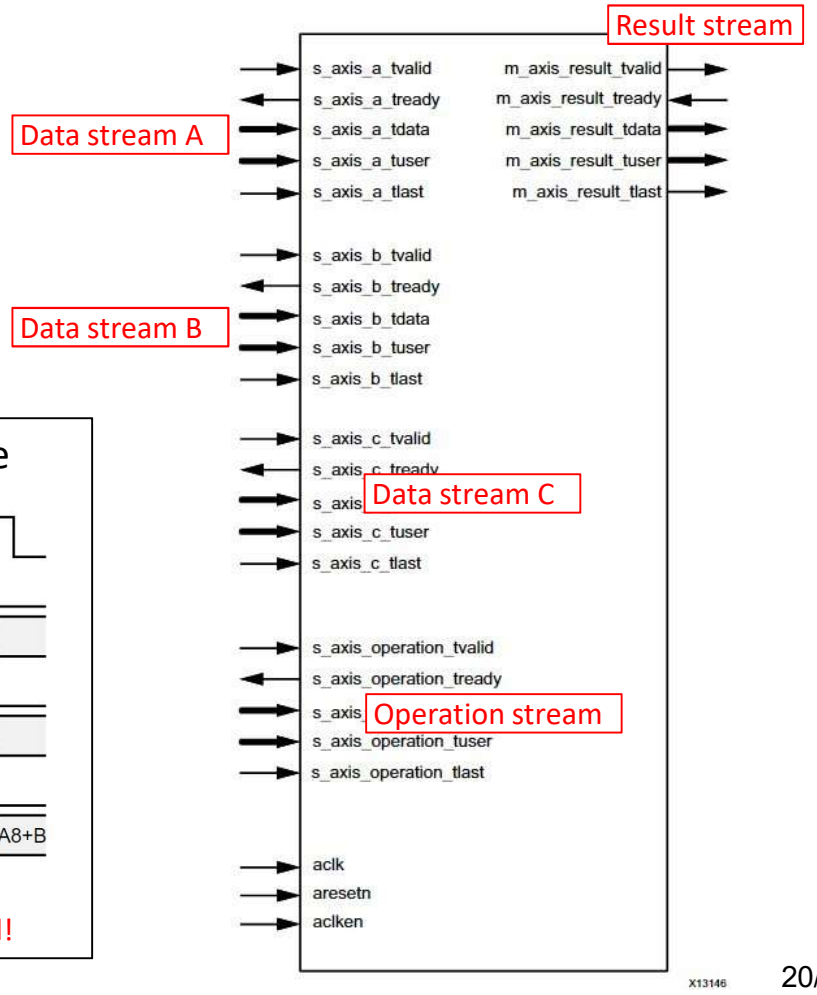
Configure the IP

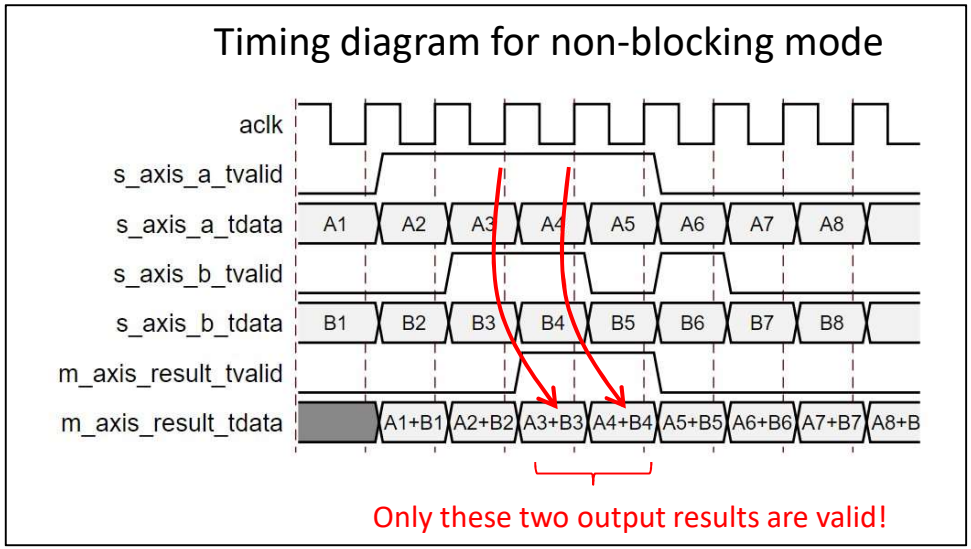
- Double-clicking the IP, a pop-up dialog will show:



AXI Floating-Point IP Interface

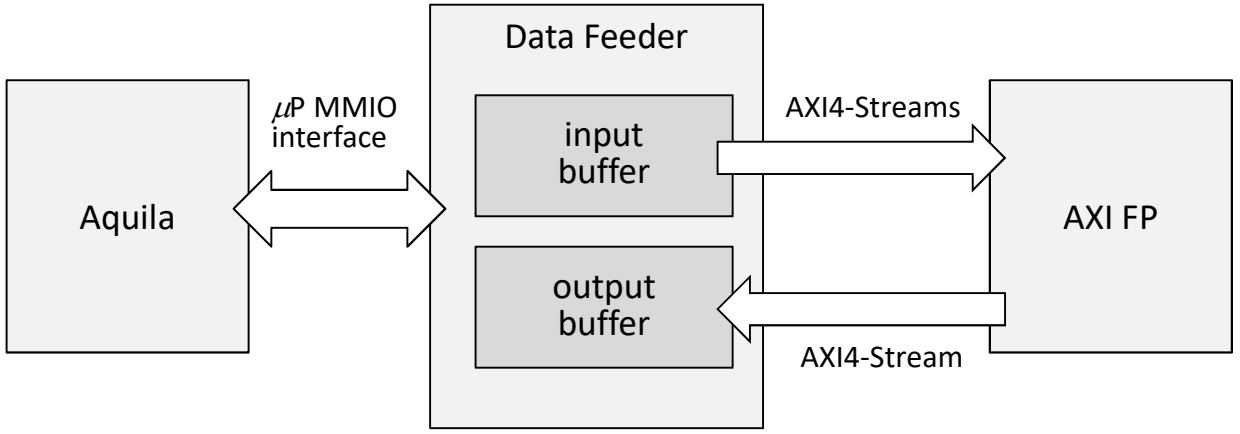
- The IP is designed to handle a sequence of floating-point operations
- Two signaling modes:
  - Non-blocking uses strobing
  - Blocking uses hand-shaking





Interface between Aquila and HW FP

- You can design a data feeder module between the Aquila and the AXI FP:
  - On the Aquila-side, use `memcpy()` to copy the vectors into the input buffer, and read the final value from the output buffer
  - One the AXI FP side, use AXI4-stream bus to send in/read out the data



Comments on the Homework

- The key point of this homework is to learn how to integrate an AXI accelerator to speed up computations
- For the accelerated system, the bottleneck is in the feeding of data streams to the AXI FP IP
  - You should measure the time spent on data feeding, and the time spent for computations, respectively
- You only need to write a detail report for this HW; there will be no demo
- You still need to upload your code to E3
  - TA will build and make sure your code actually runs