# HW 5 – Domain Specific Accelerator

沈昱宏, 110705013

***Abstract*—In this homework assignment, I explore the integration of a domain-specific accelerator (DSA) into Aquila to enhance the computational performance of a Convolutional neural network (CNN). I incorporated a DSA using memory-mapped I/O (MMIO) and AXI Stream as the on-chip bus protocol. The IP I used is the floating-point IP that help to calculate the multiply and add calculations in the forward propagation of CNN.**

***Keywords—Aquila, IP, Convolutional Neural Network***

## I. INTRODUCTION

### *A. MMIO*

Memory-Mapped I/O (MMIO) is a widely used communication model that facilitates interaction between a processor and devices within a system. In the MMIO model, specific memory addresses are reserved and mapped to the internal control registers of devices, enabling the processor to control devices by reading and writing to these addresses. This approach simplifies data transfer and control, making it efficient for operations involving devices like domain-specific accelerators.

In the Aquila system, MMIO is implemented in the soc_top module. The system memory is segmented to map devices such as UART, SPI, and DSA into distinct address spaces. Address decoding ensures that requests from Aquila's I/O ports are routed to the correct device. The MMIO process enables seamless communication between Aquila and the DSA device, triggering the floating-point IP for computations and relaying results back to the processor efficiently. Several signals are important in the MMIO model (I use the variable name in soc_top.v)

- dev_strobe

  Indicates that the master has initiated a transaction.

- dev_addr

  Specifies the target memory address for the transaction. The address determines which device is used as slave.

- dev_we

  0: Read operation (data is retrieved from the device).

  1: Write operation (data is sent to the device).

- dev_be

  A bitmask that specifies which bytes in the word are active during the transaction, we do not use this variable in this assignment because we are accessing a float.

- dev_din

  Represents the data being written to the device by the master during a write operation.

- dev_ready

  A signal from the device that indicates it is ready to complete the current transaction (e.g., it is ready to accept data for a write operation or has data available for a read operation).

- dev_dout

  Contains the data being returned by the device to the master during a read operation.

### *B. Floating point IP*

I used the floating point IP under fused multiply and add mode with single precision. I use the unblocking mode to make it easier to control. There are six input signals, s_axis_[abc]_tready and s_axis_[abc]_tdata. The IP will compute A*B+C and output the result in m_axis_result_tdata and m_axis_result_tvalid. I give all the input ready signals 1, and give the data signals the corresponding value.

### *C. Data Feeder*

I created several registers to store the value received through the MMIO interface and send them to the floating point IP. For input signals, I used the address to identify whether aquila is calling the floating point IP. In soc_top.v, when the address starts with 0xC4, dsa_sel will be set as true, allowing the data feeder to know that it is being called by controlling the strobe signal.

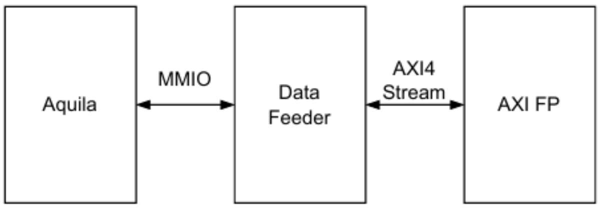

Fig. 1 Interaction between Aquila & FP

## II. Result

I optimized the conv_3d() function by setting the pointers to the correct address and reading and writing to the pointer. Since I did not modify the execution of the code, the accuracy of the 20 images is 95%.

TABLE I. Time for Execution

| **Column** | ***Time (msec)*** | ***Speed up*** |
|---|---|---|
| Original | 21244 | 1 |
| Context Switch Cycle Count | 4206 | 5.05 |

## III. Conclusion

This report integrates a domain-specific accelerator (DSA) into Aquila to enhance CNN performance using memory-mapped I/O (MMIO) and AXI Stream. A floating-point IP and a data feeder were employed for fused multiply-add operations, reducing processor workload during forward propagation. The implementation demonstrates the effectiveness of DSAs in accelerating neural network computations, paving the way for further optimization in complex systems.