Domain-Specific Accelerator

Chih-Hsuan Chen, 110550029

***Keywords*—DSA; accelerator; acceleration circuit; IP; floating point; neural networks; AI**

I. Introduction

A general CPU is designed for general purpose, and it is sufficient for most common situation. However, if the CPU is used for some specific purpose, to improve the performance, it has to provide some special instructions to do some special computations, or, add additional acceleration circuit to help the computations. While the former is more convenient, the latter usually has better performance, and it is widely used in application processors (APs) nowadays.

In this assignment, an domain-specific accelerator (DSA) with Xilinx floating-point operator IP will be intergrated to Aquila. It is aim to accelerate the computation of an MLP neural network.

II. Observation and Design

There is a DSA module added into the Aquila SoC, and the module contains a the Xilinx floating-point operator IP, which is set to return the result of *a*b+c* in one cycle (where *a, b, c* are floating point number). The data will be sent to this DSA if the accessed memory address is in 0xC4000000-0xC4FFFFFF. In more detail:

- **0xC4000000-0xC4000FFF**: Feeding the values of neurons to the DSA
- **0xC4001000**: Feeding the weights to the DSA
- **0xC4002000**: Requeseting the inner product from the DSA
- **0xC4003000**: Notifying the DSA the vector size at the beginning of the computations of each layer

A. The Values of Neurons

When computing the inner product of the vector of the previous layer neuron values and the vector of corresponding weights in the same layer, the vector of previous layer neuron values is always the same, so it only needs to be sent to the DSA once for each layer.

When each layer starts to compute, the vector of the previous layer neuron values and its size will be sent to the DSA. There is a BRAM (*BRAM_A*) in the DSA to store these neuron values. Besides, a number 1 will be appended after this vector to multiply the bias.

B. The Weights

Since all the weights are sent to the DSA exactly once originally, so it cannot be optimized by reducing the redundancy as stated above. However, the operations of data feeding and inner product calculation are independent, so they can be done

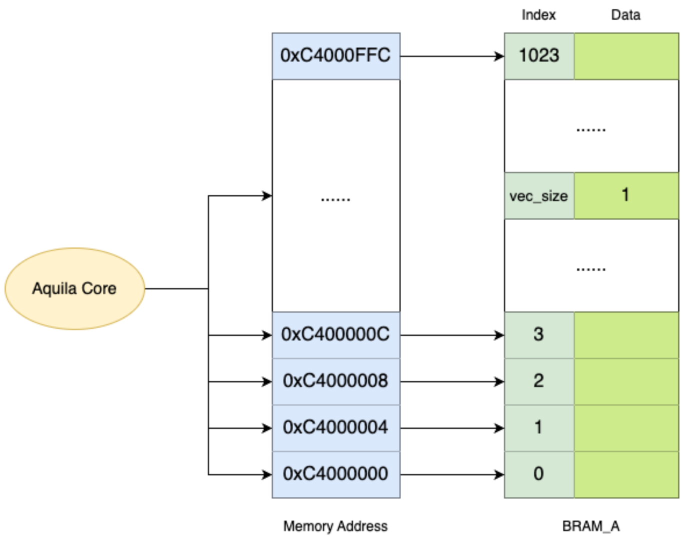


Fig. 1. The address mapping of Aquila and BRAM_A

simultaneously as the operands of inner product calculation have been fed into the DSA.

There are two BRAMs (*BRAM_B1* and *BRAM_B2*) to store the weights, one receives the next weight and the other provides the current received weight to the floating-point operator IP to compute the inner product.

To determine which BRAM receives data and which provides data, there is a variable *ping_pong* that will change its value when the DSP receives a weight. When the value of *ping_pong* is 0, the DSA stores the new weight in *BRAM_B1* and uses the weight stored in *BRAM_B2* to compute; when the value of *ping_pong* is 1, *BRAM_B1* and *BRAM_B2* change their roles.

Since each weight will be used only once, so both of *BRAM_B1* and *BRAM_B2* only have to store one number as the new weight can directly overwrite the old one.

C. Floating-Point Operator IP

The action of this IP is to compute ***a*b+c***, where *a, b, c* are floating point number. Except *aclk* and *aresetn*, the IP has four inputs *s_axis_a*, *s_axis_b*, *s_axis_c* and *s_axis_operation* and one output *m_axis_result*, each has two signals *tvalid* and *tdata*. The variable *vec_ele_idx* is used to specify the index of the next weight in the vector.

- ***s_axis_a***: Connecting to *BRAM_A*. Since *vec_ele_idx* will be incremented at the next cycle when a new weight or the request of result inner product is coming, the DSA uses *vec_ele_idx* + 1 to index the neuron values at the cycle that the DSA receives a new weight, so that the required neuron value can be prepared in one cycle.

- ***s_axis_b***: Connecting to *BRAM_B1* or *BRAM_B2*, depends on the value of *ping_pong*.
- ***s_axis_c***: Connecting to the output buffer.
- ***s_axis_operation***: The data value is always 000000 to perform addition operation. In this DSA design, the valid signal of this channel indicates whether all the data of channel *a, b, c* (whose valid signals are always set to 1) is prepared and the IP should do computation. And because the DSA design guarantees the data of channel *a, b, c* can be prepared in one cycle, the valid signal of this channel will be 1 at the next cycle when a new weight is coming, and keep for only one cycle.
- ***m_axis_result***: The result will be stored to the output buffer when the DSA receives the next weight, so the data of channel *c* will prepare its data at the same time as channel *a, b*; or, if the computation of the inner product has ended, which means that no next weight will come, the result will be stored to the output buffer when the valid signal of this channel becomes 1.

The design guarantees that the result can be calculated at the second cycle after the DSA receives a new weight.

D. Process

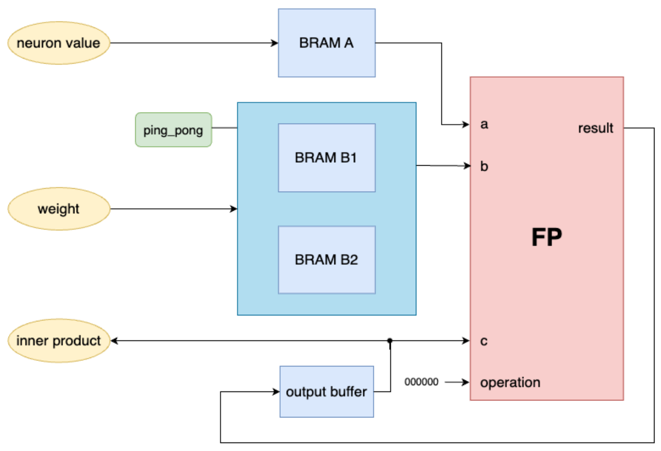


Fig. 2. The dataflow of the DSA

For each layer:

1. The process sends the number of neurons in previous layer to 0xC4003000 to notify the DSA the vector size. The DSA appends a number 1 to the end of the vector of neuron values for multiplying the bias.
2. The process sends the values of neurons in previous layer to memory address from 0xC4000000, and the DSA stores the values in corresponding address in *BRAM_A*.
   For each neuron:
   3. The process sends the weights to 0xC4001000. Each time the DSA receives a weight, it writes the weight to *BRAM_B*{*1,2*}, prepares the data for channel *a, b, c* valid signal of the floating-point operator IP, reset the output buffer if it's the first weight and reset the ready status of the DSA if it is 1, and then switch the value of *ping_pong*.
   4. If the computation of the inner product of the current neuron is ended, the DSA stores the result to the output buffer and set the ready status of the DSA to 1.
5. The process read the inner product from 0xC4002000.

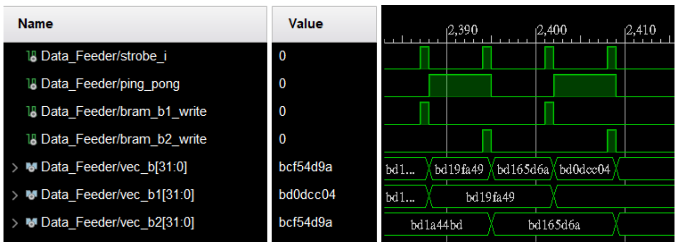
III. Result

A. Performance

This DSA reduced the time for computing from 22137 ms to **1926 ms**. The computation has been accelerated by a factor of **11.5**.

B. Ping-Pong Buffer

Fig. 3. shows that the ping-pong buffer design exactly let *BRAM_B1* and *BRAM_B2* take turns to receive and provide the weight data.

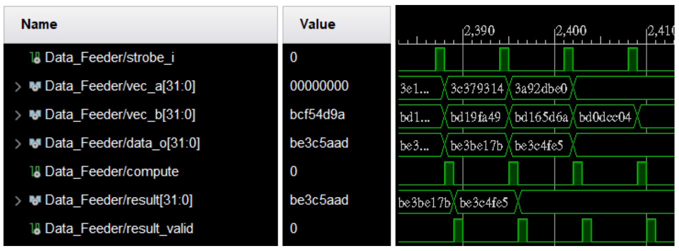


Name	Value
Data_Feeder/strobe_i	0
Data_Feeder/ping_pong	0
Data_Feeder/bram_b1_write	0
Data_Feeder/bram_b2_write	0
Data_Feeder/vec_b[31:0]	bcf54d9a
Data_Feeder/vec_b1[31:0]	bd0dcc04
Data_Feeder/vec_b2[31:0]	bcf54d9a

Fig. 3. The waveform of the ping-pong buffer

C. Data Preparing

Fig. 4. shows that the data of channel *a, b, c* will be prepared in one cycle after the new weight coming, and thus the actions of weights feeding and inner product calculation can perfectly overlap. (*compute* signal is the valid signal of *operation* channel in the floating-point operator IP.) Fig. 3. shows that the ping-pong buffer design exactly let *BRAM_B1* and *BRAM_B2* take turns to receive and provide the weight data.



Name	Value
Data_Feeder/strobe_i	0
Data_Feeder/vec_a[31:0]	00000000
Data_Feeder/vec_b[31:0]	bcf54d9a
Data_Feeder/data_o[31:0]	be3c5aad
Data_Feeder/compute	0
Data_Feeder/result[31:0]	be3c5aad
Data_Feeder/result_valid	0

Fig. 4. The waveform of the floating-point operator IP

D. Analysis

Most of the redundancy of data feeding and waiting of the data are solved in this DSA design. However, it is found that the DSA receives a weight every **7 cycles**, but it delays for **45 cycles** every 4 weights. This phenomenon is due to the cache size and cache miss, and this is the bottleneck of the acceleration.

IV. Discussion & Future Work

This assignment shows the impact and necessity of DSA. As the amount of data that the processors have to process become more and more nowadays, the performance of processors has also become a critical concern. The DSA in this assignment uses the techniques such as integrating some IPs, removing redundancy, making some actions overlap.

This DSA design still has some redundancy in data feeding. The values of neurons in the hidden layers are evaluated by the DSA but not read from the given file, so they can be stored in another BRAM as their values have been calculated (have to consider the activate function if it is used). Such design can be implemented with the concept of ping-pong buffer, and it can let the DSA get the values of the neurons after the first layer in one cycle. This neural network uses ReLU as the activate function, which is simple enough to be implemented with circuit (use the MSB to determine the sign of the number). But the number of neurons after the first layers is not many, so this design may not show significant improvement.

The other way to further improve the speed of the neural network is to modify the floating-point precision of the weights, to let the length of each weight be 16 bits. Since the entry length of the cache is fixed (128 bits), this modification can let each cache entry be able to store 8 floating-point numbers, to reduce the influence of cache miss, which is mentioned above that limits the improvement by the DSA.