ECSE 541 Assignment2 Report

Hung-Yang Chang[1]
[1]Student ID:260899468, Department of Electrical and Computer Engineering, McGill University

I. Introduction

THIS work implemented hardware-software cooperation system with bus interface and memory to enable matrix multiplication.

II. Design Detail

This system comprises of four operation blocks: two computation blocks **Hardware**, and **Software**. One communication block **Bus**, and one storage block **Memory**. Matrix calculation is done by cooperation between software and hardware, then stored back into memory. Meanwhile, Bus consists of two interfaces serving as the communication bridges between software, hardware, and memory.

The detailed computation flow lists as follows. First, Memory starts initializing matrix a, b, and c value, which matrix A and B will be read, and matrix C will be modified into computation result later by software and hardware. Once initialization is finished, computation will begin from software by first write zero to matrix C in Memory to perform zero-initialization. In this step, the software serves as master writing Matrix C, and Memory serves as minion receiving zero. Afterward, the software utilizes *Software_call_Hardware* function to call hardware to cooperate to decrease reading and writing time during computation. In this step, software serves as a master and hardware serves as a minion. Once hardware receives cooperation requests from software, it serves as master reading Matrix A and B from memory and stored into hardware register. Therefore, now hardware serves as a master and memory serves as a minion to provide matrix A and B. Matrix multiplication is then computed in hardware then writes the multiplication result into memory. So far this is one computation loop, and This loop will run 1000 times. One computation loop flow is summarized in Table I.

For each operation block, more details lists as in the following subsection.

A. Memory

Memory utilize 1D vector *MEM[MEM_SIZE]* to represent 2D matrix A, B, and C, and MEM_SIZE is equal to 108. In other word, MEM[0] - MEM[35] stores matrix a, MEM[36] - MEM[71] stores matrix b, and MEM[72] - MEM[107] stores matrix c. Memory has a thread called *Memory_read_write()* which sensitive to clock positive edge keeps checking if there is any request from master. If both required address is valid and operation is read or write, memory will start serving as minion and acknowledge master. Depending on the require operation, memory will send out read data if master request reading or memory receive write data if master request writing.

Step	Master	Minion	Operation
1	Software	Memory	Initialize Matrix C with zero
2	Software	Hardware	Request SW/HW cooperation
3	Hardware	Memory	Read Matrix A from memory
4	Hardware	Memory	Read Matrix B from memory
5	Hardware	Memory	Write multiplication result to memory

TABLE I
COMPUTATION FLOW WITH MASTER-MINION PAIR

B. Software

Software has a thread called *Software_cal* which sensitive to clock positive edge. In *Software_cal*, 1000 times matrix multiplication is realized. As mentioned in computation flow, for each loop, software first does zero-initialization then and then call hardware to cooperate to perform matrix multiplication. The reason calling hardware to cooperate is that if the computation is performed in the software, for each output element, 6 reads from matrix A and another 6 reads from Matrix B would be needed. Since 36 output elements, the software would need to perform 36*12 = 432 reads. However; for hardware, it only requires 72 reads for multiplication.(Check Section II-C). Since now matrix multiplication is moved to hardware, wait() should be added to *Software_cal* in order to wait enough time until hardware finishes the computation.

C. Hardware

Hardware has registers that can hold data and those data can be reused. In the software, 432 reads are needed to finish the matrix multiplication. To target maximizing reuse, each data has to be read only once from the memory. Since there are a total of 72 elements in the two matrices (A and B). Therefore, two registers A with length 36 for matrix and matrix B is implemented in hardware so as to hold data and reusing it. In this way, only 72 reads are needed to perform.

Hardware has two threads: *Hardware_listen_to_software* is sensitive to clock positive edge keeps checking if there is any request from master(software), and *Hardware_cal* is also sensitive to clock positive edge doing matrix multiplication when software request hardware to cooperate. Once hardware receives cooperation request from software, it starts reading Matrix A and B from memory and stored into hardware registers, and then perform c[i][j] += a[i][k] * b[k][j] by using three for-loops. When the computation is finished, hardware will and write results back to memory. In this work, the hardware throughput is 3 (216 multiplications / 72 reads). While if using software to do matrix multiplication, its throughput is 0.5 (1 multiplication / 2 reads). Therefore, hardware throughput is 6x improvement as compared to software.

However, minimum throughput is still not achieved. As Mohamed Abdelgawad stated in the team, broadcasting matrix B column to MAC units in a specific order can even push throughput to 6 (36 multiplications / 6 reads), which is 12x improvement as compared to software. This could be an improved version of the current implementation to be work on.

D. Bus

Bus consists of two interfaces (master and minion) serving as the communication bridges between software, hardware, and bus. Bus has one thread called *Bus_arbiter*, which is sensitive to clock positive edge keeps checking if there is any valid master request.

For master, *Request* checks if the master id is valid to decide whether store request information or not. *WaitForAcknowledge* keeps waiting until the master id is correct or acknowledgment is valid to continue work. For reading and writing, FIFO queue is utilized to enables *WriteData()* to let new data be inserted in the queue in a different position based on when it was last granted. *ReadData()* will then always check the first request pending in the queue by reading the buffer front. Moreover, wait() is inserted here to prevent the master from reading from an empty queue.

For minion, *Listen()* will be called from minion (either memory or hardware) to get operation information from the master. *Acknowledge()* will then let the master get acknowledgment that master will start to process its request. Given that only one master and one minion can be paired simultaneously, resetting operation information other than master id to default value is performed here so that the bus minions would not acknowledge the current master component more than once. for *SendReadData*, and *RecievedWriteData()* have similar manner as described in master read and write, and again wait() is inserted in SendReadData to prevent sending data from an empty queue.

III. Timing annotation

Given the condition of how many cycles for each operation takes, fine-grained timing annotation is performed in this implementation by adding proper numbers of ”wait(Clk.posedge_event())” to proper operation as requirement as follow: request takes 2 cycles, acknowledge takes 1 cycle, write takes one cycle, and read take two cycles. Moreover, because of the fact software is calling hardware to cooperate, software needs to wait until hardware finish calculation. Therefore, ”wait(194 *clock_period , SC_NS)” is added to software right after software calling hardware to cooperate. 194 cycle is actually equal to (76 + 76 + 40 + 2) cycles which will be further explained in Section IV.

IV. Performance

For the throughput, as declared in Section II-C, software calling hardware to perform matrix multiplication will have three times improvement compared to software performing calculation itself. For the execution time, the total time to finish the simulation is 1585083 nanosecond, which is around 238000 cycles. In other words, it takes 238 cycles to finish each iteration. More specifically, 40 cycles are needed to initialize c to zero in software (4 cycles for software to request memory and get acknowledge, then 36 cycles are needed to write into the matrix), then 4 cycles for software to call hardware to cooperate. Afterward, it takes 76 cycles for each to read out matrix A and B and store in the register. (For each read, 4 cycles for hardware to request memory and get acknowledge, then 72 cycles are needed to read matrix out) Another 40 cycles are required to write matrix C back to memory. Finally, 2 additional cycles is added to re-invoke iteration. Although matrix multiplication solely in software is not implemented in this project. It can still be expected execution time for such hardware-software partition in system-level design would be much lower because registers in hardware can be reused frequently to save the execution time to frequently read data.

V. Acknowledgment

I would like to thank Mohamed Abdelgawad for sharing design ideas in teams, Jack Hu for explaining in detail how to implement the bus, and also other classmates for helping me out. Last but not least, much thanks to professor for giving me extra time to make this work more completed!

Appendix A

Bus Arbiter and simulation result is shown in next page.

```
void Bus_arbiter()
{
  while (true)
  {
    wait(Clk.posedge_event());
    // if  (current.op != MODE_None && current.len >= 0)
    if (current.len >= 0)
    {
        current.master_id = masterbuffer.master_id;
        current.address = masterbuffer.address;
        current.op = masterbuffer.op;
        current.len = masterbuffer.len;
        length_sent = 0;
        length_recieved = 0;

        // print out mode other than MODE_None
        if (current.op != MODE_None)
        {
          cout << " @" << sc_time_stamp() << " Master-Minion connected now... Master id: " << current.master_id
          << " (0SW, 1HW) Address: " << current.address << " Operation mode: " << current.op << "(0W, 1R, 2C) length: " << current.len << endl;
        }
    }
  }
}
```

Fig. 1. Bus Arbiter Code

```
@0 s 1-time Calculation...
@0 s Start doing software initalization....
@9990 ps Master rquest now.... Master id: 0(0SW, 1HW), Address: 72, Op: 0(0W, 1R, 2C) , Length: 36
@9990 ps Software waiting acknowledgedment
@9990 ps Master-Minion connected now... Master id: 0 (0SW, 1HW) Address: 72 Operation mode: 0(0W, 1R, 2C) length: 36
@16650 ps Master-Minion connected now... Master id: 0 (0SW, 1HW) Address: 72 Operation mode: 0(0W, 1R, 2C) length: 36
@23310 ps Address is valid, Memory served as minion now...., operation: 0 (0W, 1R)
@23310 ps Software get acknowledgedment
@263070 ps Software finsihed initalization......
@263070 ps Software serves as master calling hardware to cooperate......
@276390 ps Master rquest now.... Master id: 0(0SW, 1HW), Address: 2000, Op: 2(0W, 1R, 2C) , Length: 1
@276390 ps Software waiting acknowledgedment
@276390 ps Master-Minion connected now... Master id: 0 (0SW, 1HW) Address: 2000 Operation mode: 2(0W, 1R, 2C) length: 1
@283050 ps Master-Minion connected now... Master id: 0 (0SW, 1HW) Address: 2000 Operation mode: 2(0W, 1R, 2C) length: 1
@283050 ps Hardware serves as minion and listen to software (master) now
@289710 ps Hardware acknowledged software now
@289710 ps Hardware and software are cooperating now... reading A and B now....
@289710 ps Hardware served as master reading data from memory now......
@303030 ps Master rquest now.... Master id: 1(0SW, 1HW), Address: 0, Op: 1(0W, 1R, 2C) , Length: 36
@303030 ps Hardware waiting acknowledgedment...
@303030 ps Master-Minion connected now... Master id: 1 (0SW, 1HW) Address: 0 Operation mode: 1(0W, 1R, 2C) length: 36
@309690 ps Master-Minion connected now... Master id: 1 (0SW, 1HW) Address: 0 Operation mode: 1(0W, 1R, 2C) length: 36
@316350 ps Address is valid, Memory served as minion now...., operation: 1 (0W, 1R)
@316350 ps Hardware recieved acknowledgedment...
@329670 ps Hardware reading....1
@795870 ps Hardware reading....36
@795870 ps Hardware Read from memory successfully....
@795870 ps Reading Register A successfully
@795870 ps Hardware served as master reading data from memory now......
@809190 ps Master rquest now.... Master id: 1(0SW, 1HW), Address: 36, Op: 1(0W, 1R, 2C) , Length: 36
@809190 ps Hardware waiting acknowledgedment...
@809190 ps Master-Minion connected now... Master id: 1 (0SW, 1HW) Address: 36 Operation mode: 1(0W, 1R, 2C) length: 36
@815850 ps Master-Minion connected now... Master id: 1 (0SW, 1HW) Address: 36 Operation mode: 1(0W, 1R, 2C) length: 36
@822510 ps Address is valid, Memory served as minion now...., operation: 1 (0W, 1R)
@822510 ps Hardware recieved acknowledgedment...
@835830 ps Hardware reading....1
@1302030 ps Hardware reading....36
@1302030 ps Hardware Read from memory successfully....
@1302030 ps Reading Register B successfully
@1302030 ps Hardware finsihed matrix multiplication!
@1302030 ps Hardware served as master writing data to memory now......
@1315350 ps Master rquest now.... Master id: 1(0SW, 1HW), Address: 72, Op: 0(0W, 1R, 2C) , Length: 36
@1315350 ps Hardware waiting acknowledgedment...
@1315350 ps Master-Minion connected now... Master id: 1 (0SW, 1HW) Address: 72 Operation mode: 0(0W, 1R, 2C) length: 36
@1322010 ps Master-Minion connected now... Master id: 1 (0SW, 1HW) Address: 72 Operation mode: 0(0W, 1R, 2C) length: 36
@1328670 ps Address is valid, Memory served as minion now...., operation: 0 (0W, 1R)
@1328670 ps Hardware recieved acknowledgedment...
@1335330 ps Hardware writing....1
@1568430 ps Hardware writing....36
@1568430 ps Hardware write to memory successfully......
@1568430 ps Hardware finsih writing back to memory!
@1568430 ps C value is
0, 0, 0, 0, 0, 0, 0, 162, 222, 219, 207, 218, 0, 284, 474, 416, 443, 483, 0, 76, 129, 115, 111, 139, 0, 48, 101, 83, 95, 101, 0, 130, 209, 199, 196, 220,
```

Fig. 2. Simulation detail for first iteration