Integer MULT – Project Report

Hua, Mengli

Li, Junlei

Metkar, Piyush Vijay

Mu, Xingyu

Vyas, Dhwani Rakesh

Xiao, Yi

Zhu, Tianyi

October 24th, 2018

CSE 7381: Computer Architecture

Southern Methodist University

Author Note

Mengli Hua, Junlei Li, Piyush Vijay Metkar, Xingyu Mu, Dhwani Rakesh Vyas, Yi Xiao and Tianyi Zhu are now students in Computer Science, Southern Methodist University.

Contact: [mhua@smu.edu](mailto:mhua@smu.edu), [junleil@smu.edu](mailto:junleil@smu.edu), [pmetkar@smu.edu](mailto:pmetkar@smu.edu), [xingyum@smu.edu](mailto:xingyum@smu.edu), [drvyas@smu.edu](mailto:drvyas@smu.edu), [yixiao@smu.edu](mailto:yixiao@smu.edu) and [zhut@smu.edu](mailto:zhut@smu.edu).

**General Project Description**

Using Verilog, expand the design of the basic ALU of our previous Verilog homework assignment to handle integer multiply operations. Add appropriate registers and other hardware as needed. Assume 32-bit two’s-complement numbers will be used, same as for the current ADD and SUB operations. Test using both positive and negative integer numbers.

**Specific Implementation Plan**

Algorithm thinking： Shift addition

The algorithm can be divided by three parts: First multiply the binary multiplicand and each bit of the multiplier to obtain the partial product which has the same number of digits as the multiplier. Secondly, adding the partial product according to the weighted value and then add the partial product compression. Last, A super forward bit adder module is usually used. The design of the multiplier is naturally concentrated on the performance optimization of each module to achieve the purpose of fast speed and small area.

**High-level algorithm:**

**Python:**

def multiply ( multiplicand, multiplier ):

result = 0

for i in range(32):

if multiplier & 0x1 == 1: // if the i-th bit of multiplier is 1, then add multiplicand to result

result += multiplicand

multiplicand = multiplicand << 1 // multiplicand shift left 1 bit

multiplier = multiplier >> 1 // multiplier shift right 1 bit

return result

C:

signed int multiply( signed int multiplier, signed int multiplicand ) {

signed int product = 0;

while (multiplier != 0) {

if ((multiplier & 1) != 0) {

product = product + multiplicand;

}

multiplier = multiplier >> 1;

multiplicand = multiplicand << 1;

}

return product;

}

**Hardware Components used:**

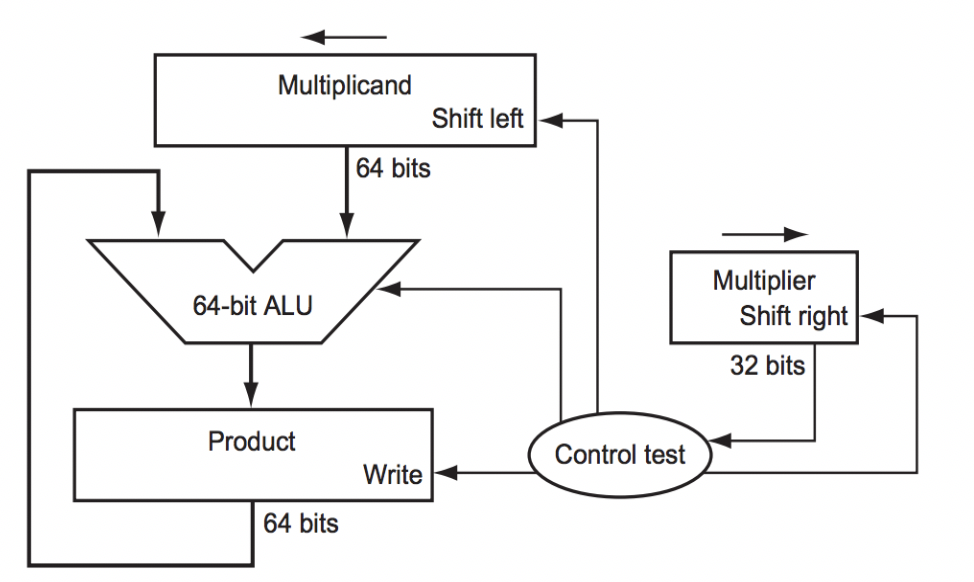


Figure 1. First version of the multiplication hardware

According to the Figure 1(please see Appendix A for first version of hardware), we can understand how the first version of hardware works, the hardware had been drawn to resemble the method of paper-and-pencil, which shows that the data flows from top to bottom. In order to understand this method clearly, we assume the multiplier is in the register with the size of 32-bits, and the 64-bit product register is initialized to 0.

There is no doubt that we should move the multiplicand from right to left for one digit in each step, since it has likely to be added to the intermediate products. A multiplicand with 32 bits would move from right to left when it passes over 32 steps, in this case, a Multiplicand register with 64 bits is needed, and this Multiplicand register should be initialized with 32 bits in right side and 0 bit in the other side. After that, this register is then shifted left 1 bit each step to align the multiplicand with the sum being accumulated in the 64-bit Product register (Patterson, D. A., Hennessy, J. L,2015).

As Figure 2 mentioned (Please see Appendix B for the first Multiplication Algorithm), we can find there are three basic steps needed for each bit. The least significant bit of the multiplier (Multiplier0) determines whether the multiplicand is added to the Product register. The left shift in step 2 has the effect of moving the intermediate operands to the left, just as when multiplying with paper and pencil (Koren. 2010).

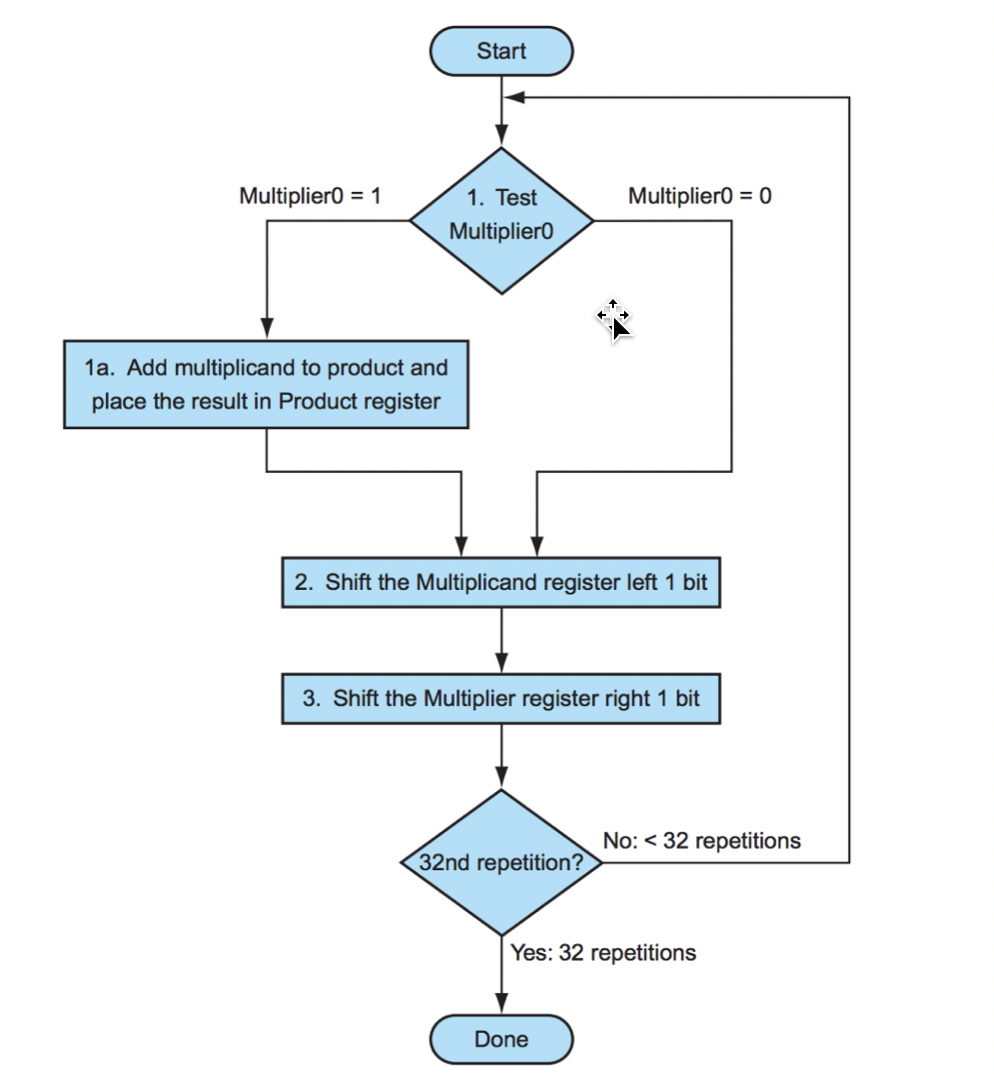
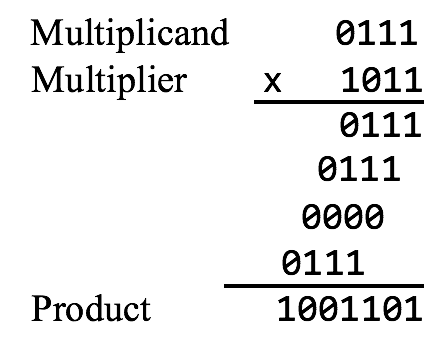


Figure 2. The first Multiplication Algorithm

**Method to handle signed integers (Positive integers and Negative integers):**

Because the numbers we accept are two’s complement numbers, to handle multiplication for both positive and negative integer numbers, we reverse the sign of negative numbers by taking the two’s complement operation for their binary representation. Then we get the positive values for the negative integer numbers. Therefore, we can continue doing the multiplication by using shift-add method which was discussed previously. At the last step, we take the XOR operation on the leading bit of the original values of both inputs to determine if the sign for the final product is positive or negative. That is, if one input has a leading bit with “0” and another input has a leading bit with “1”, after taking XOR operation, we will get “1” as the leading bit for the final product which represents a negative number. If both of them have the same leading bit “0” or the same leading bit “1”, we will get “0” as the leading bit for the final product which represents a positive number. Because the multiplication used two positive values, we will definitely get a positive temporary result. We will need to take the two’s complement operation for the temporary result to make it negative to get the correct final product if we determined that the final product should be a negative number after the XOR operation taken before. Otherwise, the final product would be the temporary result.

**Example for the shift-add multiplication for unsigned integers: (7\*11=77)**

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Product | Multiplier | Multiplicand | Operation |
| 0 | 0000 0000 | 1011 | 0000 0111 | Initialization |
| 1 | 0000 0111  0000 0111  0000 0111 | 1011  1011  0101 | 0000 0111  0000 1110  0000 1110 | Add Multiplicand to Product  Shift left Multiplicand 1 bit  Shift right Multiplier 1 bit |
| 2 | 0001 0101  0001 0101  0001 0101 | 0101  0101  0010 | 0000 1110  0001 1100  0001 1100 | Add Multiplicand to Product  Shift left Multiplicand 1 bit  Shift right Multiplier 1 bit |
| 3 | 0001 0101  0001 0101 | 0010  0001 | 0011 1000  0011 1000 | Shift left Multiplicand 1 bit  Shift right Multiplier 1 bit |
| 4 | 0100 1101  0100 1101  0100 1101 | 0001  0001  0000 | 0011 1000  0111 0000  0111 0000 | Add Multiplicand to Product  Shift left Multiplicand 1 bit  Shift right Multiplier 1 bit |

The example above shows the processes of shift-add multiplication for unsigned integers. For this example, we accept two 4-bits positive integer numbers which are 7(0111) as the multiplicand and 11(1011) as the multiplier and finally get an 8-bits product of 77(0100 1101). The reason that the multiplicand register has 8 bits here is because it need to move from right to left when it passes over 4 steps as discussed before. In each step, if the last bit of the multiplier is 1, we need to add the current multiplicand to the current product. Then we shift the multiplicand left 1 bit and shift the multiplier right 1 bit. Otherwise, if the last bit of the multiplier is 0, we only need to shift the multiplicand left 1 bit and shift the multiplier right 1 bit. When the value of multiplier become 0, we are done.

**Test Plan:**

After all other works done, which means the hardware and software designs are ready. Then the test can be processing with the MARS and Verilog, to ensure the Verilog and MIPS code can run correctly, respectively.

To test MIPS code, we can give values directly to the code, and generate results to identify whether the code is successful. If failed, by a list of test set, it seems possible to have the solution to correct the code.

For the Verilog code, we can create a new Verilog file as the testbench. The function of a testbench is to apply inputs to the design while testing, and then produce the outputs in a readable and user-friendly format. All the Verilog we plan on using in our hardware design must be synthesizable, meaning it has a hardware equivalent. The Verilog we write in a test bench does not need to be synthesizable because we will only ever simulate it (Justin Rajewski, January 10, 2018).

After tests has done, we can use the test results to see whether the hardware and software design is valid and try to find the way to correct or improve them.

### **Milestone and Deliverable**

* Oct. 24, 2018: First time meeting. Discussion about the basic duties for each member.
* Oct. 25, 2018: All designers discussion on their specific implement plan. Each part implement plan draft need to be finished and given to writer.
* Oct. 26-27, 2018: Writer completes the proposal draft. All members read and given suggestions.

*Deliverable:* ***Proposal***

* Nov. 10, 2018: Hardware designers complete hardware design. Tester ready to do hardware test.
* Nov. 15, 2018: Hardware function has been correctly completed. Writer prepare to start intermediate status report.
* Nov. 16-17, 2018: Intermediate status report finish.

*Deliverable:* ***Hardware function (Verilog code, a .v document), Intermediate status report***

* Dec. 1, 2018: Software designers complete software design. Tester ready to do software test.
* Dec. 5, 1028: Software function has been correctly completed. Writer prepare to start final report writing.
* Dec. 6-8, 2018: Final project report finish.

*Deliverable:* ***Software function (MIPS code, a .asm document), Final project report***

### **Team Duties**

Project Manager:

As the manager of this project, I will make the tasks plan with milestone and deliverable for the whole team. Also, I will always pay attention on the project processing to make sure that everyone is catching up to the schedule by complete their specific tasks on time. In addition, I will help the writer to notify other designers to give the information he needs to complete all the documents. The last specific duty of mine is to communicate with “upper management” and let all the members know the information given by Dr. Manikas.

Technical Writer:

The main duty for a technical writer is to develop proposal, report drafts, including final draft. I would pay more attention to how to deliver the information in a clear, concise and correct way to reader. I choose APA Formatting and Style guide [5] as format to write this paper. I would also double check every teammates’ paper before adding them to reports, making sure the information they deliver is correct.

Hardware Designers:

As a hardware designer, I will develop the Verilog code and handle the component designs for the project. I will also write clear comments in my code to make it easier for teammates to understand and develop tests.

Software Designers:

As a software designer, we will be in charge of designing and developing the MIPS code to implement and run the Integer MULT operations on hardware. We will ensure that code developed by us would properly integrate with the hardware developed in Verilog, as well as fix any potential issues found during testing. We will also strive to maintain high standards in our coding work by writing well commented code. The project will be divided in weekly targets and we would work as per that.

Test Engineer:

Test is a very important part to ensure the reliability of the code we design. Test Engineer will do the test after software and hardware design completed, which means it is the last part of coding work. So, it is necessary to remind the schedule of the whole project as early as possible and other members’ work process. To test the validity of their MIPS and Verilog code. The main duty is to design methods and codes to test MIPS and Verilog code respectively. In MIPS field, initial plan is to directly assume values with the MIPS code to ensure its correctness. On the other hand, Verilog testing needs to create testbenches to verify the design functionality of hardware design, and we can see the result in the Verilog console. After coding work done, test engineer should give suggestions and take measures to correct or improve the functionality of the software and hardware code.

Assignment:

|  |  |  |
| --- | --- | --- |
| **No.** | **Roles** | **Name** |
| **1** | Project Manager | Hua, Mengli |
| **2** | Technical Writer | Li, Junlei |
| **3** | Hardware Designer(s) | Xiao, Yi, Zhu, Tianyi |
| **4** | Software Designer(s) | Metkar, Piyush Vijay, Vyas, Dhwani Rakesh |
| **5** | Test Engineer | Mu, Xingyu |

# Reference

Patterson, D. A., Hennessy, J. L. (2015). Computer organization and design: The hardware/ software interface. Retrieved October 25, 2018, from <http://www4.comp.polyu.edu.hk/~comp2421/ComputerOrganizationAndDesign5thEdition2014.pdf>

Koren. (2010). Part 4: Datapath Design – Multiplication and Floating-point representations and operations. Lecture. Retrieved October 25, 2018, from <http://euler.ecs.umass.edu/ece232/pdf/04-MultFloat-11.pdf>

Baruch, Z. F. (2002). Structure of computer systems. Cluj-Napoca: U. T. Pres. Retrieved October 25, 2018, from <http://users.utcluj.ro/~baruch/book_ssce/SSCE-Shift-Mult.pdf>

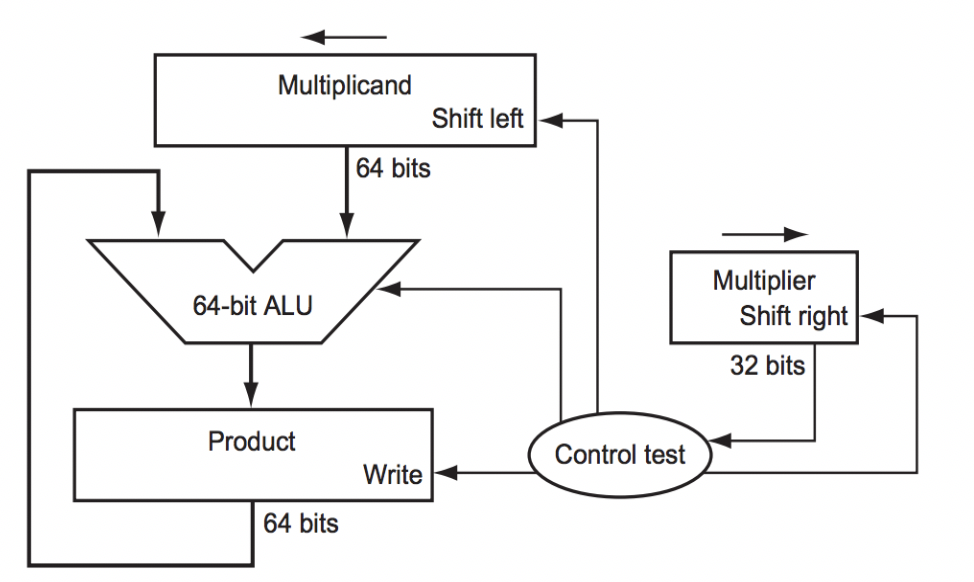
Justin Rajewski, *Writing Test Benches*, January 10, 2018

Purdue Online Writing Lab. *APA Formatting and Style Guide* Retrieved from

<https://owl.purdue.edu/owl/research_and_citation/apa_style/apa_formatting_and_style_guide/general_format.html>

Appendix A

First version of the multiplication hardware



Appendix B

The first Multiplication Algorithm

