Lab 4: Bit-Pair Recoded Multiplier

ECE327: PROJECT 4

November 17, 2019

Bradley Selee
Clemson University
Department of Electrical and Computer Engineering
bselee@g.clemson.edu

Abstract

In Project 4 a multiplier was created using the Bit-Pair Recoding Algorithm. This multiplier takes up to 8 bits, 4 bits on the FPGA, and multiplies them. They may either be negative or positive. The reason bit-pair was a good algorithm of choice is because it will always take $\frac{n}{2}$ partial products to complete the multiplier, where n = number of bits in the multiplicand. This is not always true in Booth's algorithm; in some cases, however, Booth's may be quicker. Each multiplication takes $n + 2$ clock cycles. Upon the last clock cycle, the register B contains the lower bits of the output, while register C contains the upper bits of the output. These registers combine to form the entire product [2].

1 Introduction

Project 4 recreates the bit-pair multiplication algorithm. Similar to lab 3, many individual components were created to perform each step in the algorithm; then, the components were instantiated and connected with signals to create the entire circuit. The circuit required four registers, each performing different functions, two multiplexers, an adder, and a mealy finite-state machine (FSM). The block diagram shown in the lab manual was slightly modified for the top-level entity, and some signals, such as the clock and reset, were added to the final design, which are not shown in the block diagram.

Each component of the lab required a test bench, this verifies that each component works properly, ultimately reducing the number of errors. Once each component was verified, the components were instantiated and connected with signals to create the entire bit-pair algorithm. The final circuit was verified in a test bench to show various multiplications and its output. Finally, the circuit was wrapped in a board file to map the functionality to the FPGA. Once the project was uploaded to the FPGA, a final test was run to ensure the board produced the proper multiplication that the user requested.

2 Components

The bit-pair circuit designed in lab 4 required several different components. There consisted of 4 registers, an adder, two multiplexers, and a control FSM. This can be seen in the figure below:

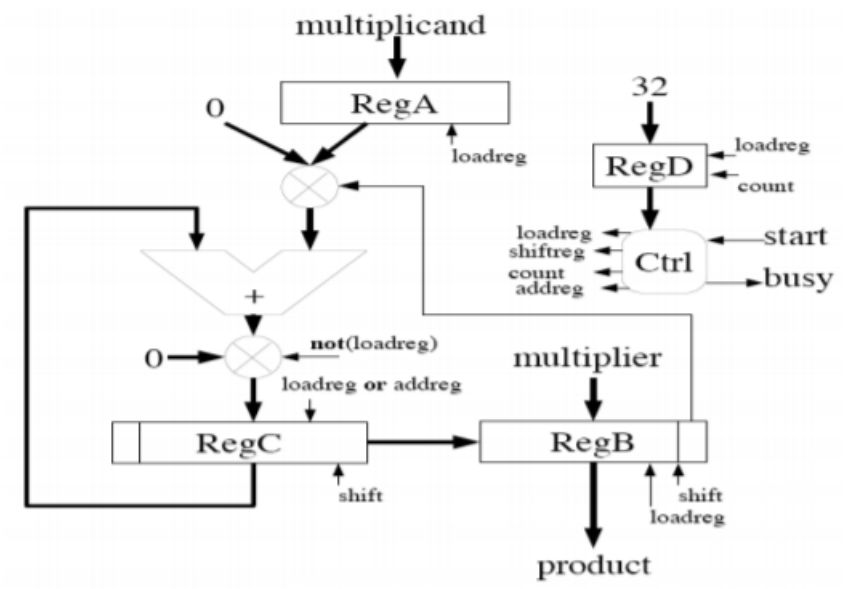

Figure 2.1: Lab 4 Bit-Pair Circuit [2]

2.1 Registers

Register A takes a generic 8-bit multiplicand. Its job is to recode the multiplicand so that the proper recoding can be selected by the multiplexer. Because there are 5 different recodings, register A outputs 5 different 9-bit values, 9 bits to account for the left shift in 2 of the recodings. The recoding scheme is shown below

Binary	**Decimal**	**Description**
000	0	Zeroes
001	1	Multiplicand
010	1	Multiplicand
011	2	Multiplicand and left shift
100	-2	Two's complement and left shift
101	-1	Two's complement
110	-1	Two's complement
111	0	Zeroes

Register B takes a generic 8-bit multiplier. This registers job is to send the 3-bit binary value to the multiplexer, so it may select the proper recoding from the table above. Register B also stores the lower bits for the final product. Every other clock cycle, 2 bits are stored inside register B, while the multiplier is simultaneously shifted out.

Register C takes a 1-bit carry and a 9-bit recoded input. This register shifts the lower 2 bits onto its but to register B, then sign extends the upper bits, applying the carry if necessary. This new value after the shift is then sent to the adder which creates the next partial product.

Register D takes only takes two 1-bit enables. The first enable loads the number of partial products, $\frac{n}{2}$, and the second decrements the $\frac{n}{2}$. The register outputs the current value of $\frac{n}{2}$ so that after $\frac{n}{2}$ partial products, the FSM can reach the *First* state.

2.2 Multiplexers

The 5-1 multiplexer selects the recoding based on the table above, then outputs it to the adder. The recoding bits are given by the multiplier in register B.

The 2-1 multiplexer selects a 9-bit 0 vector during the load state. After the load state, the multiplexer selects the addition from the adder. During the load state, register C must output the first partial product which is why the 2-1 mux initially outputs 0's.

2.3 Adder

The adder is used to add the partial product with the recoding. Typically, a ripple-carry adder is used to add and subtract binary numbers; however, VHDL is smart enough

to perform this mechanism in the background. As shown on page 229 of the text book including the library [1]

```
use ieee.std_logic_unsigned.all;
```

This Allows the use of the + and - operators to perform the arithmetic. Furthermore, in bit-pair, one must account for the carry; this adder outputs the carry of the addition and sends it to register c to later be used

2.4 Control Unit FSM

The state machine controls the entire system. Due to the nature of the bit-pair circuit, the state depends on both the inputs and outputs, a mealy type state machine is the most logical and efficient state machine to use. Because mealy fsm's are typically less states than moore types, the processor FSM only required 4 states. This includes a and initial state, a load state, an add state, and a shift state.

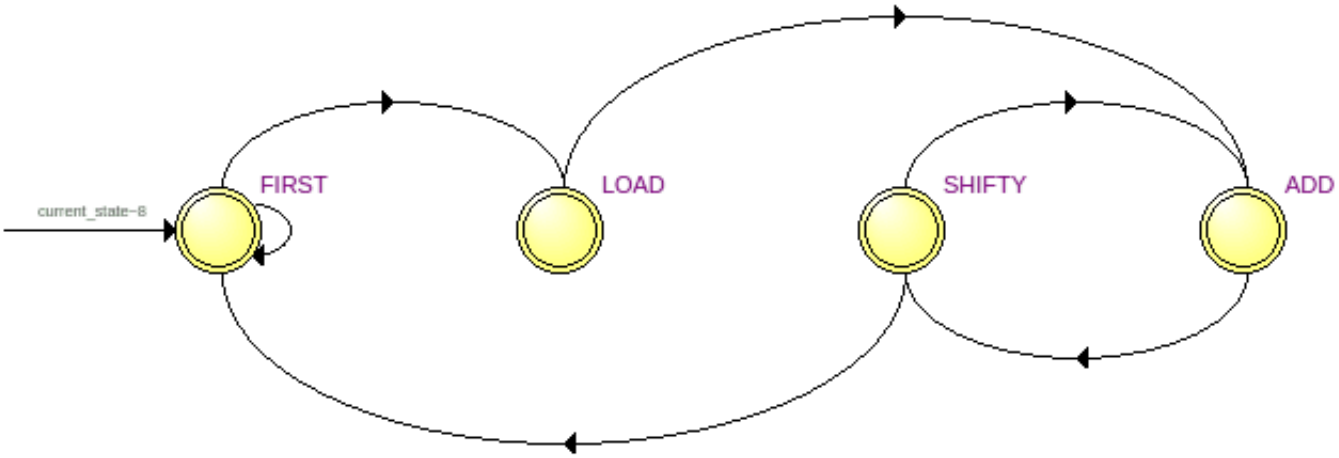

Figure 2.2: Quartus Generated Mealy State Machine

The FSM takes a 1-bit start, 1-bit reset, and a 3-bit partial product count for the inputs. The FSM outputs the register enables,a busy signal and a done signal. The state machine was based off a loading state, then each state in the bit-pair algorithm as shown in figure 2.2.

3 Testing and Verification

Once each component was created for the bit-pair circuit, it was crucial to ensure that each component worked as designed. This was accomplished by creating test benches for each component, then verifying the proper output was displayed on the wave form. Below shows the wave forms of each component

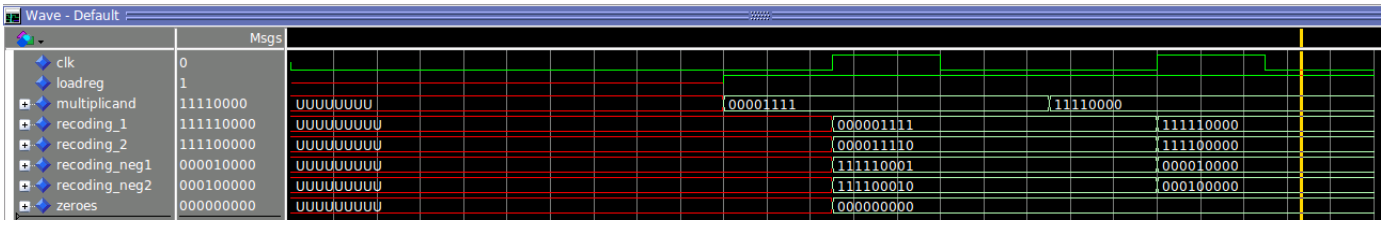

Figure 3.1: Register A waveform

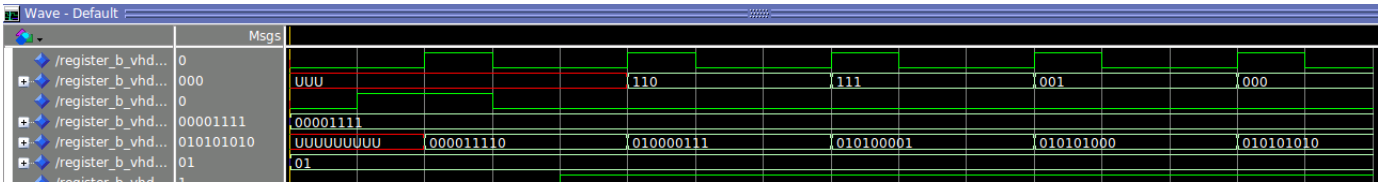

Figure 3.2: Register B waveform

Figure 3.3: Register C waveform

Figure 3.4: Register D waveform

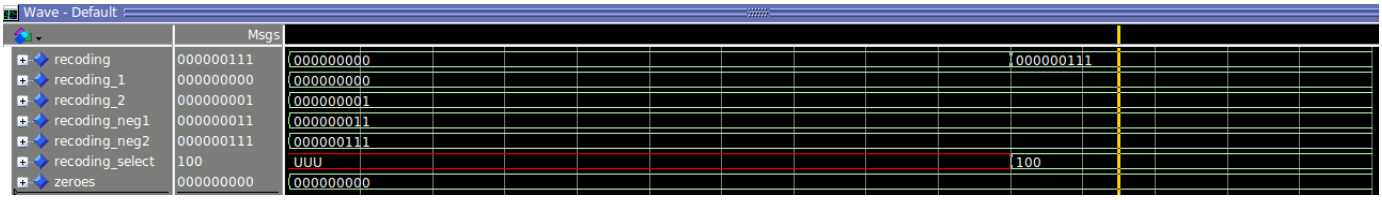

Figure 3.5: 5-1 Multiplexer

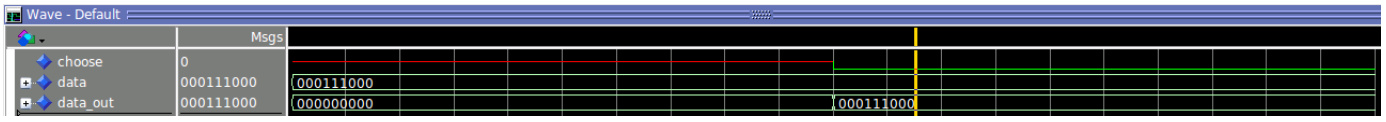

Figure 3.6: 2-1 Multiplexer

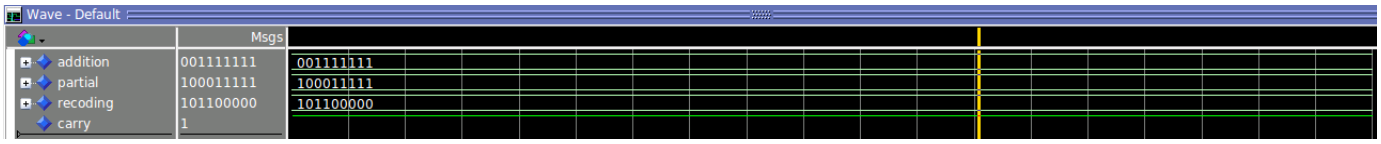

Figure 3.7: Adder

4 Bit-Pair Circuit

Once all of the individual components were created and verified, the components were connected through signals to form the bit-pair algorithm.

4.1 Architecture

Within the processor VHDL file, the I/O ports were created for the system. The bit-pair circuit has inputs of the multiplicand, multiplier, and the final product for the output. Then, each component was instantiated and port mapped to each other.

4.2 Testing and Verification

Once all components were mapped, Quartus was used to generate the block diagram to ensure that the components were properly connected show in figure 4.1. The final circuit was tested thoroughly with a test bench as shown in (figure 4.2). This waveform displays the necessary values to ensure functionality such as the states, multiplicand, multiplier, the recoding, and the final product. This example demonstrates $-18 \cdot 55 = -990$.

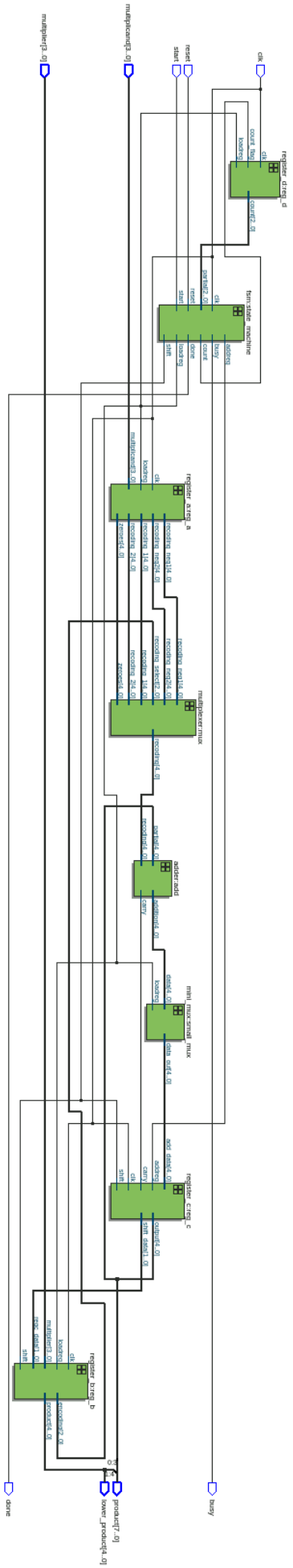

Figure 4.1: Complete Bit-Pair Circuit

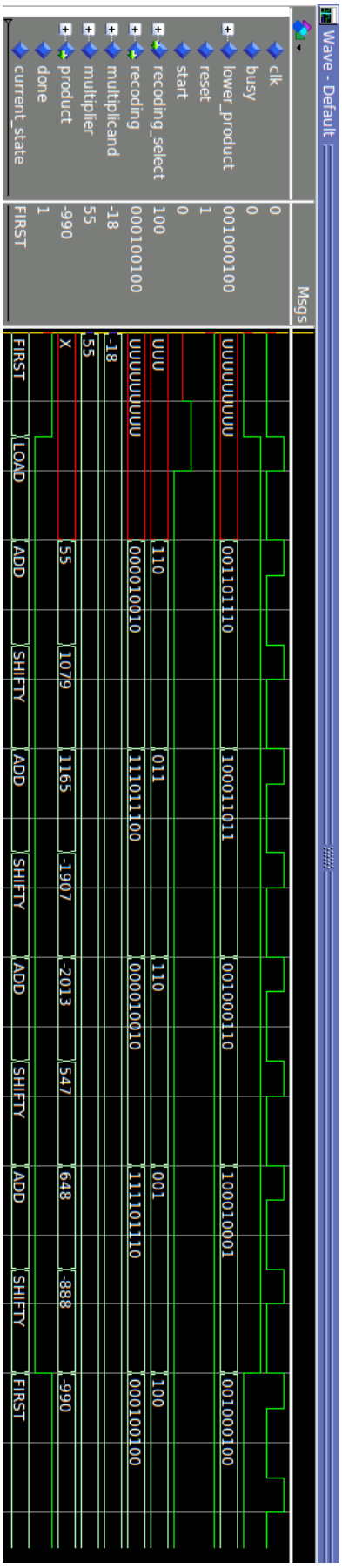

Figure 4.2: Bit-pair circuit waveform demonstrating functionality

5 Discussion and Results

Lab 4 was conceptually the hardest lab. In order to implement a fully functional multiplier using the bit-pair algorithm, it was necessary to completely understand every step to this algorithm. Essentially, a component was created for each step in the bit-pair algorithm, then everything was connected and the multiplier worked. Unlike the processor, even though each component may have worked individually, when combined the circuit may not have been functional. Also, running the circuit was much easier; the values were loaded and the circuit was *"powered on"*, then after $n + 2$ clock cycles, the output was displayed. The multiplier begins in the *FIRST* state until the *start* input is high, thus the *busy* signal also becomes high. Then, the circuit loads each register in the *LOAD* state, including the recoding based on the lowest three bits in the multiplier. After a clock cycle, the circuit begins its first add in the *ADD* state. Once the addition is complete, the FSM transitions to the *SHIFT* state. In this state, the lower two bits within register C shift into register B to begin the product. Simultaneously, the bits stored in register B get shifted out two bits at a time. Every shift, register D decrements its count, $\frac{n}{2}$, and after four shifts, the FSM transitions back to the *FIRST* state, where the *done* signal becomes high and the final product is displayed.

6 Conclusion

This lab allowed me to take a real world problem, multiplication, analyze it, and understand how to implement it with a digital circuit. Because of the complexity of the first lab, I was able to understand the purpose of the registers much easier in lab 4 and did not get stuck very often, this reasoning is also the same for the FSM. I believe this lab was successful because my circuit was able to perform several multiplications, including negative numbers, and display the proper output. If I was to redo this lab, I would have started earlier so I could test its functionality further and possibly display some values on the hex display.

References

[1] W. J. Dally, R. C. Harting, and T. M. Aamodt. *Digital Design Using VHDL: A Systems Approach*. Cambridge University Press, Cambridge, 2015.

[2] M. Smith. *Lab 4: Lab 4: Bit-pair Recoded Multiplier*. Clemson University, 2019.

7 Appendix

No further references were needed for this project.