ECE 3270

LAB 4 REPORT

June 13, 2022

Tim Driscoll
Clemson University
Department of Electrical and Computer Engineering
tdrisco@clemson.edu

Abstract

The main goal of this lab is to implement a Bit-Pair Recoded fixed point multiplier. The multiplier was implemented using a combination of registers, an adder, a multiplexer and a general control unit. The general control unit was designed using a Mealy implementation of a finite state machine. The modular components were combined to create a structural top level entity that represents the Bit-Pair Recoded multiplier.

1 Introduction

Lab four consists of a modular design with eight different parts that were combined to develop the top level entity. The top level entity that was created is a Bit-Pair recoded multiplier. In the design there were three basic and generic components that were designed and implemented, these components included a basic adder, 5-to-1 multiplexer and 2-to-1 multiplexer. The 5-to-1 multiplexer is used to select which version on the recoded multiplicand is used, the adder is then used to perform the series of adds implemented in bit-pair multiplication. The 2-to-1 multiplexer was the used to determine what is loaded into a register within in the design. The design also included four different registers (A-D) that all performed different functions. The A register is used to load and store the multiplicand as well as recode it in accordance to what is neede for bit-pair multiplication. Register B was used to initially load in the multiplier, it is then used to simultaneously used to shift in the final product and shift out the pair of bits needed to determine the recoding of the multiplicand. In the end register B will store the lower half of the final product. Register C is then used to hold the partial products during the computation, it shifts bits into the B register and provides an input into the adder. Lastly register D is used to represent a simple counter, used to determine the number of shifts needed to perform the computation. Lastly, in order to tie the whole design together a control unit was created and implemented as a mealy finite state machine. The control unit was to assert all the necessary enable bits and correctly set some of the outputs of the circuit. All of these components were then combined to create a final circuit which could carry out the process of Bit-Pair recoded multiplication.

2 Design

2.1 Multiplexer Implementation

A multiplexer is another relatively simple logic device that was implemented twice in this lab. A multiplexer is used to output a single piece of data from many inputs. The input is determined by selector where each input has a corresponding selection code. The first of the two multiplexers was a 5-to-1 multiplexer which used a three bit selector. The selector was represented by the pairs of bits from the multiplier as well as a dummy bit. These bit pairs were then used as the selector to determine which recoded version of the multiplicand was to be used based on the transitions. The encoding scheme can be observed in Table 1, and is representative to each group of bits relation to a string of ones. Four of the inputs to the 5-to-1 multiplexer come from register A which outputs all the recoded versions of the multiplicand. The last input isn't defined as an input but is simply coded to be all zeros. The output from this register is directed as one of the inputs into the adder. The second multiplexer was a very simple 2-to-1 implementation. The select was represented by a single bit (1 or 0) where a one represented one input and a zero represented the other input. The select bit is tied to the loadreg signal which is asserted by the control unit. The inputs to the 2-to-1 multiplexer come from the sum of the adder and an input simply representing all zeros. The output is directed into the

C register.

Table 1: Select Input Correspondence

Select Bits	Output
000	0x Multiplicand
001	+1x Multiplicand
010	+1x Multiplicand
011	+2x Multiplicand
100	-2x Multiplicand
101	-1x Multiplicand
110	-1x Multiplicand
111	0x Multiplicand
Other	0x Multiplicand

2.1.1 Test of Multiplexer

The multiplexer design was based off of the multiplexer that was previously designed in lab 1 and 3. The basic functionality of the multiplexer was already confirmed and the only adjustments that were made involved the encoding scheme and the altering of the number of inputs. Through the test of the final circuit implementation the multiplexers functionality could be confirmed through the observation of the internal signals.

2.2 Adder Implementation

The adder unit is a simple logic device that is implemented in process. The adder unit would simply function by adding the two inputs together. This was simpler design then what was used in lab three because it eliminated the complexity of needing subtraction capabilities. For the implementation used for this lab the IEEE numeric_std library was used. This library allowed for casting of the inputs as signed values. After both inputs were cast as signed the operation could performed. Lastly the value that was obtained from the operation (output) needed to be casted back to a STD_LOGIC_VECTOR. The implementation of this adder did not involve a carry out bit and was simply truncated to the width of the output bits. Any needed sign extension that was needed in the implementation is handled by the C register during the shifting process.

2.2.1 Adder Testing

In order to confirm the functionality of the adder unit a test bench was created to confirm functionality. This test bench is represented by Figure 1. In order to test the functionality with the test bench a series of four different adds were completed. These four adds were specifically used because they will represent the four adds implemented in the final circuit when the multiplicand is "10111010" and the multiplier "10010110". Using these specific adds will help verify the functionality of the adder as well as help verify the final

circuit. The results from these adds were verified by hand a proved the correct designed functionality of the adder implemented in the bit-pair recoded multiplier.

2.3 Register A Implementation

Register A was created using a simple D flip-flop that included an enable bit and involved some extra functionality. A D flip-flop is a relatively simple logic component that is used to update data from the input to the output based off of the rising or falling edge from a clock and the assertion of an enable bit. When the desired clock edge is detected and the enable bit is high the D flip-flop latches, setting the current input to the output. If the input is changed it will not update the output until the desired clock edge is received. In the lab the D flip-flop was implemented to latch on a rising edge if the enable bit was set high. This was done using a simple if statement that checked for a rising edge and the value of the enable bit. If both the constraints were detected then the input would be assigned to the output. The extra component of register A was that it had four different outputs that would all be update based on a single input and and the previously stated conditions. In the design of the bit-pair recoded multiplier the input of register A was designed to represent the multiplicand. Register then functioned as a way to store the multiplicand but also a way to output and store the four different recoded versions of the multiplicand. These four different recoded versions of the multiplicand represent the multiplicand times 1, -1, 2, and negative two these four numbers also make up the outputs of register A.

2.3.1 Test of Register A

The testing that went into the register was relatively low level and abstract. Since the register was implemented after the D flip flop that was designed in lab 3 the basic functionality was already confirmed. Although this is true the added functionality of more outputs that all had a different recoded representation of the input needed to be tested. This test included a simple test bench that can be represented by Figure 2. This test bench is designed to show how a simple input is handled by register A and how the different outputs are represented. Through the test bench it can be confirmed that the DFF latches on a rising clock edge when the enable bit is high. Then by looking at the outputs and checking by hand it can be confirmed that all four recoded versions of the multiplicand were correctly determined and outputted. Further testing of register A was done in the testing of the final circuit. The success of the final circuit further confirmed the correct implementation of register A.

2.4 Register B Implementation

Similar to the description of register A in section 2.3 the blue print of register B was based upon the original D flip flop that was designed in lab 3. The design included an extra enable bit that indicated register B should perform the added functionality. The initial functionality of the B register is to store the value of the multiplier input. The

multiplier gets latched to the output when the first of the two enable bits is asserted (loadReg) and there is a rising clock edge. The added functionality of register B was to be able to shift in two new bits in the most significant bit position, and shift out the least significant bits. Register B was the width of the data plus one allowing for B to store the size of the data multiplier and single dummy bit. The dummy bit gets initialized to 0 when the multiplier is loaded into the register. Following the initialization with each sequential shift the dummy bit gets set to the bit in the first position of the B register. The bits that get shifted into the B register are represented by the two least significant bits in the C register, these two bits also represent the lower 8 bits of the final product. Starting from the first shift and moving to the last shift the two bits shifted in represent the least significant bits (0 and 1) in the product to the 6th and 7th bits in the product. After each shift the three least significant bits in the register represent the next bit-pair and the dummy bit. These three bits also represent the select bits for the 5-to-1 multiplexer and are used to determine which recoded version of the multiplicand is sent to the adder. In summary the B register stores and holds multiplier initially then shifts out the multiplier as bit-pairs while simultaneously shifting in the lower half of the product.

2.4.1 Test of Register B

Similar to register A the main functionality of the D flip flop with an enable was already confirmed. The focus of this test was to confirm the added functionality of shifting bits in and out was properly working. The test bench for register B can be referenced by looking at Figure 3. This test bench starts by latching an a bit value (multiplier) to the output (q). As shown in the test bench this latching occurs when the enableBit is held high and there is a rising edge on the clock. It can also be shown that during this latch the initial dummy bit of 0 is added to the input. After this there is a series of two shifts completed by the register. The enableBit is set low and the shiftEnable bit is set high, the shifts then occur on the next to rising edges of the clock signal. The first shift shifts in the bits "00" which results in the output transitioning from "100111000" to "001001110". For the second shift in bits are "11" which results in the output transitioning from "001001110" to "110010010". This test bench shows and proves the functionality of the B register.

2.5 Register C Implementation

Similar to the description of register A in section 2.3 the blue print of register C was based upon the original D flip flop that was designed in lab 3. Also similar to register B as described in section 2.4 the extra functionality of the C register was to be able to handle shifting bits. Specifically, register included three different enable bits. The first two enables (enableBit1, enableBit2) had the same functionality which was to enable the input to be latched to the output on a rising clock edge. The reason there were two enable bits is because one enable was used to initialize the register and load in zeros from the 2-to-1 multiplexer. Whereas when the second of the two enables was asserted it represented loading in the sum of the adder from the 2-to-1 multiplexer. Then when the

shiftEnable was asserted the register functioned by shifting out the two least significant bits from the register and assigning them as an output. When the two bits are shifted out the register was sign extended to shift in the correct leading bits and assign the correct output. Lastly the value that is stored in the C register is the partial product during the computations. Upon completion of the multiplication the C register stores the upper half of the final product. In summary the C register is used to store the partial products during the computation and shift these values into the B register in order to combine and create the final product.

2.5.1 Test of Register C

Similar to register A and B the main functionality of the D flip flop with an enable was already confirmed. The focus of this test was to confirm the added functionality of shifting bits in and out was properly working. The test bench for register C can be referenced by looking at Figure 4. This test bench starts by asserting the first of the two enable bits to latch a value to the output ("110011100"). The enable bit is then set low and the shift bit is asserted. Upon the next rising clock edge the two least significant bits get shifted out "00" and assigned to the shiftOut output. The original output is then sign extended resulting in two ones to be shifted in and the output to be updated to "111100111". After this is completed the shiftEnable is brought low and the second of the two enable bits is asserted and a new input is latched to the output "000011111". The same shift process is carried out to prove that the output is properly sign extended and updated after the shift. The final output is equal to "000000111", which correctly represents how the shift should have been handled. Through this test bench the functionality of the C register was confirmed.

2.6 Register D Implementation

Similar to the description of register A in section 2.3 the blue print of register D was based upon the original D flip flop that was designed in lab 3. For register D the functionality of the D flip flop component was the same but there is lack of a data input and the inclusion of another enable bit. There is no data input in this register because its basic functionality is to act as a counter which is implemented as a simple shift register. The data that acts as the input is an internal signal that gets initialized to a vector of zeros. The initialization of this data occurs when the enableBit is set and there is a rising clock edge. This sequence also results in a vector of zeros to be latched to the data output of the flip flop. The second enable bit (countEnable) is the bit that needs to be asserted in order for the register to increase the count. Along with count bit being asserted a rising clock edge needs to occur for the count to increase. The count is also latched to the data output during this sequence. The count works by shifting ones into the original vector of zeros. Table 2 represents the count sequence when the width of the data being multiplied is eight bits. Since the width of the data being multiplied is eight bits there will be four separate adds (width of data / 2). Register D is used to count and ensure that only (bits/2) adds occur in the overall process. Thus as shown by the example in

table 2 the count is complete when the output of register D is a vector of all ones. The output from register D is an input into the control unit that determines when overall multiplication is done.

Table 2: Register D Count Sequence

Clock Cycles	Output
0	0000
1	1000
2	1100
3	1110
4	1111

2.6.1 Test of Register D

It was important to properly test the functionality of register D using a test bench in order to ensure proper functionality. Since register D involved some larger design changes to the original D flip flop thorough testing was needed. As shown by the test bench in figure 5 the general flow of the test was to initialize the internal data signal and then prove that the count takes for cycles. Then the internal data signal was reset and the count was run through again with an additional clock cycle (5) to show that this wouldn't have any affect on the output. Further testing of register D was done in the testing of the final circuit. The success of the final circuit further confirmed the correct implementation of register D.

2.7 Control Unit Implementation

The final component that needed to be created was a control unit that was implemented as a mealy finite state machine. The control unit functioned by assessing the value of the current state and the inputs to determine how the outputs should be updated. The outputs of the control unit consisted of a group of enable bits and two different status bits. The assertions of these different would occur on the transitions between the three states in the Mealy design. The three states that were used in the design were a wait state, an add state and a shift state. The state diagram generated by quartus can be viewed below by observing figure 6. When observing the state diagram an idea of the transitions can be obtained. The control unit starts in the wait state which also represents the reset state. Once the start bit is raised high the control unit transitions to the add state. From the add state there will always be a transition to the shift state, because each add in the bit-pair design has a resulting shift. Once in the shift state if all the adds are completed then the control unit goes back to the wait state. If there are still more adds that need to occur the control until cycles back to the add state. In order for the control unit to determine the if there are any more adds or if the multiplication is done the resulting count from the D register is used. To provide a more detailed list of the outputs of the

state machine can be observed in table 3. The design of a mealy finite state machine involved two separate processes, the first process updated the current state based on the clock and the reset. The second process was to update the output and it was based on the inputs and the current state. The inputs for this machine were simply the start input and the counter that came from the D register. In VHDL the design included these two processes that were modeled very similarly. In both processes were case statements checking the current state, which all head nested if statements to determine the affects of the inputs. From this point the processes would respectively update the current state and the outputs.

Table 3: State Machine Outputs

State	Transition To	loadreg	addreg	shiftreg	count	busy	done
Wait	Wait	1	0	0	0	0	0
Wait	Add	0	1	0	1	1	0
Add	Shift	0	0	1	0	1	0
Shift	Add	0	1	0	1	1	0
Shift	Wait	0	0	0	0	0	1

2.7.1 Test of Control Unit

The control unit was initially tested through the creation of a specific test bench to cycle through all the states and check the respective outputs. The initially design was also previously confirmed in lab 3 and further confirmed in the testing of the final circuit. The specific testing of this control unit can be observed by the test bench shown in figure 7. The test bench runs through a full hypothetical add cycle although the counter was manually inputted and would not behave in this exact way. The control unit starts in the wait state an only the loadreg is asserted. Once the start bit is set high there is the output updates and loadreg is brought low and the addreg, count, and busy bits are brought high. All of these assertions can be confirmed in table 3. After transitioning to the add state it can be scene that on the next rising edge of the clock there is a transition to the shift state. This transition results in the shiftreg and busy bit to be brought and all other bits to be brought low. The control unit then cycles between the add state and the shift state until the counter represents 1111 as designed for an a bit input. Once the counter represented 1111 the done bit was asserted until the next rising edge. On this rising edge everything was cleared and the loadreg was brought high showing that the control unit was back in the wait state. Note that the current_state signal wave was added to the test bench for enhanced readability.

2.8 Final Circuit Implementation

The final step of the lab was to create the overall bit-pair recoded multiplier, that comprised of all the components described in the above sections. Since all of the

components of the final circuits were already created the final circuit was simply a structural implementation. This structural implementation involved initializing all the components, declaring instances, and mapping internal signals to connect the circuit. The only inputs that were needed for the final circuit were clock, reset, start, the multiplier and the multiplicand. Start and reset were inputted directly into the control unit. Start represented the start bit that needed to be enabled for the state machine to transition out of the wait state and reset was simply the reset for the state machine. The multiplier was inputted directly to register B and represted the value that RegB loaded in when the loadreg signal was asserted. The multiplicand was inputted directly into register A and represented the value that RegA loaded in when the loadreg signal was asserted. Lastly the clock was inputted into all the registers and the control unit to represent the system clock. The outputs of the final circuit were busy, done and the final product. Both busy and done were outputted by the control unit to represent when the circuit was in the computation process, done with the computation or waiting. Waiting was represented by neither done or busy being high. Lastly the final output represented the product of the multiplicand and multiplier, this product was represented by the output of both register C and register D. Register C represented the upper half of the partial product and register B represented the lower half of the partial product. Since all the final circuit components were modular it made for a simple final circuit implementation.

2.8.1 Test of Final Ciruit

In order to test the implementation of the bit-pair recoded multiplier in the final circuit multiple test benches were created. These test benches were used to look at a variety of different inputs and confirm the outputs. The first test bench that is shown and represented by figure 8. The multiplicand and multiplier are two negative eight bit inputs representing -70 and -106 respectively. When these to numbers are multiplied their product should equate to 7420. 7420 can be represented in binary by "0001110011111100". This product is the same number that can be observed by the product output of the final circuit when the done bit is high. By observing the left side of figure 8 it can be observed that the done bit is high and the correct number is represented by the product. In figure 9 a similar instance was verified in the same way, except the inputs represented a negative multiplicand and a positive multiplier. This test bench was specifically used to confirm that a negative output could be correctly calculated. The third test bench represented by figure 10 was used to confirm the final circuits ability to handle inputs of different widths. Since all the components in the circuit were created using generics and mapped to the single generic in the final circuit, the ability to change data widths was simple. By adjusting the generic in the final circuit to the desired data width all components would be correctly updated to the desired size. Figure 10 shows that the final circuit was able to handle a six bit input and produce the correct output. The output was verified similar to how the first output was verified. Lastly figure 11 was a simple test bench used to show implementation of a reset in the middle of a cycle (sent back to wait state), as well as the current state remaining in the wait state while the start bit is set to 0.

3 Questions

3.0.1 Describe in detail how your machine would operate on inputs with an odd number of bits.

In order to perform bit-pair recoded multiplication when the inputs are odd only requires a single adjustment. This adjustment would be to sign extend the multiplier by a single bit in order to account for the last bit-pair. Once the multiplier has been sign extended by one all the operations can be normally carried out. In my machine this would only require adjustments to the B register from how I originally designed my circuit. Since I wasn't able to specifically determine whether the input had an odd or even number of bits the solution had to be independent of this. In order to handle the sign extending I created a signal within the B register that was two bits larger then this multiplier input. By creating a signal that was two bits larger then the multiplier I could both sign extend the multiplier and include an initial dummy bit of 0. I then added an extra output to register B which was used to directly represent the 3 multiplexer select bits. This three bit output was represented by the three least significant bits of the added signal. Like the output the signal was also shifted in order to send out the correct sequential bit-pairs. Since this signal was sign extended the it would correctly handle odd inputs and the extra bit would simply be ignored for even inputs. The only other piece that would need to be handled is that for an odd input the B register would store the lower n+1 bits and the C register would store the upper n-1 bits. Whereas for an even input both registers would store an equal n bits creating for a product of 2n bits. This component change created for an implication in my design that I was unable to find a universal solution to. I was able to produce valid outputs for both even and odd inputs by adjusting where the product was obtained from for each case. Although this was true I was unable to correct this error and find a solution that would effectively work for both even and odd inputs without adjustments. An example of a valid output when the adjustments were made for an odd input is represented below by figure 12.

3.0.2 Describe in detail the modifications you would need to make to your machine if you were making a Moore design.

In order to make the current machine a Moore design the modifications would strictly occur in the control unit. As previously discussed the control unit is currently designed to represent a Mealy finite state machine. The largest difference between a Moore and Mealy finite state machine design is that a Moore machine's output is only dependent on the current state whereas a Mealy's output is dependent on both the current state and the input. Thus the output of a Moore machine is delayed by a clock cycle and requires more states. A Mealy state machine will update on the transition of states as suppose to a Moore machine which updates on the states. Thus with this being said the most major modification would be the addition of an extra state, this additional state would represent the done state. Table 3 below represents the update to the needed states and the updated outputs for each state. These outputs are no longer occurring on the

transition between states but on the states themselves. This change would have to be represented by the modification of the process in the control unit that is updated on the change in the clock or reset. Instead of the shift state either going to the add state or back to the wait state it would either go the add state or the done state. The done state would then transition back to the wait state. The next modification that would need to be changed is an update to the output process in the control unit. To represent a Moore design the output would only be dependent on the current state so the the inputs start and counter would be removed from the sensitivity list. Lastly in the output process the done state would need to be included and the outputs for each state would need to be updated to represent table 3.

Table 4: State Machine Outputs

State	loadreg	addreg	shiftreg	count	busy	done
Wait	1	0	0	0	0	0
Add	0	1	0	1	1	0
Shift	0	0	1	0	1	0
Done	0	0	0	0	0	1

4 Conclusion

The main goal of this lab was to design a final circuit that could perform bit-pair recoded multiplication with 8-bit inputs. The goal was achieved by combining modular sub-components of the circuit together in a final implementation. Each component was individually created and tested to verify its correctness prior to its addition to the final circuit. The registers,multiplexer control unit and adder were all modeled off of implementations in lab 3. The registers, adder and control unit were individually tested to verify their added and changed functionality. Through testing of these sub-components and implementation in a final circuit the final goal was achieved, and correct multiplication was performed and verified.

The lab provided a great learning experience and allowed for an application of many of the topics that were discussed during lecture. It was especially important to have a thorough understanding of bit-pair recoded multiplication. Some of the main lessons that were learned throughout this lab included how to implement a mealy version of a finite state machine in VHDL, how to create counters, expand the functionality of simple registers and how to better use modelsim to test and debug a circuit. This lab showed the effectiveness of taking time to plan out and design a modular circuit in order to work up to a larger circuit. Throughout this lab the only errors that occurred were minimal syntax bugs that were easily fixed, and the occasional issue of race conditions in the final processor test bench. The result of limited errors can be credited to the use of test benched to thoroughly verify and eliminate bugs in the sub components. Overall the lab provided a great learning experience through the process of implementing a bit-pair

recoded multiplier.

5 References

M.Smith "Digital Computer Design" Clemson University Holcombe Department of Electrical and Computer Engineering, January 2020

M.Smith "ECE3270 Digital System Design Lab 4: Bit-Pair Recoded Multiplier" Clemson University Holcombe Department of Electrical and Computer Engineering, January 2020

W. Daily, R. Curtis Harting and T. Aamodt, Digital Design Using VHDL a systems approach. Cambridge University Press

6 Appendix

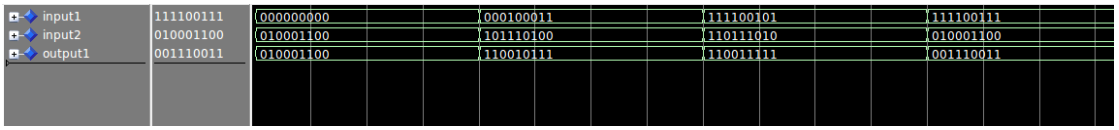

Figure 1: Adder Test Bench

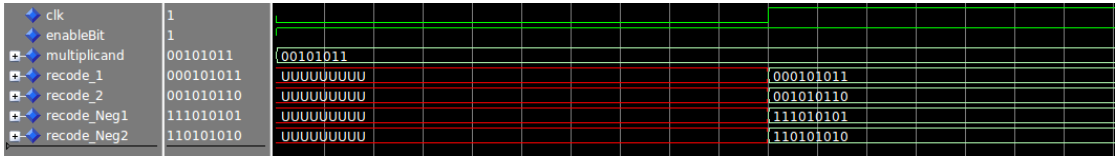

Figure 2: Register A Test Bench

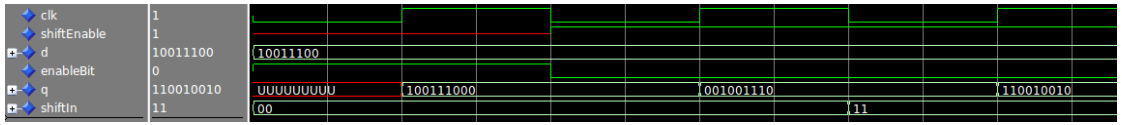

Figure 3: Register B Test Bench

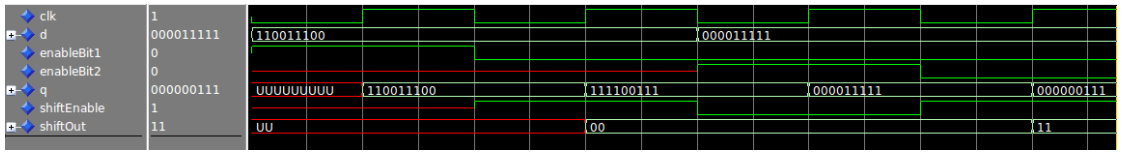

Figure 4: Register C Test Bench

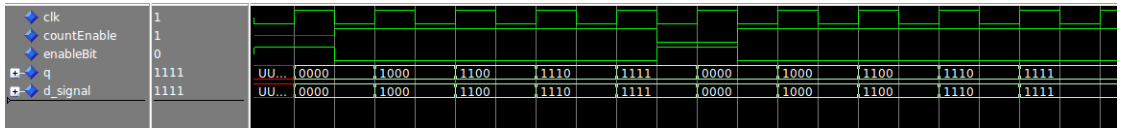

Figure 5: Register D Test Bench

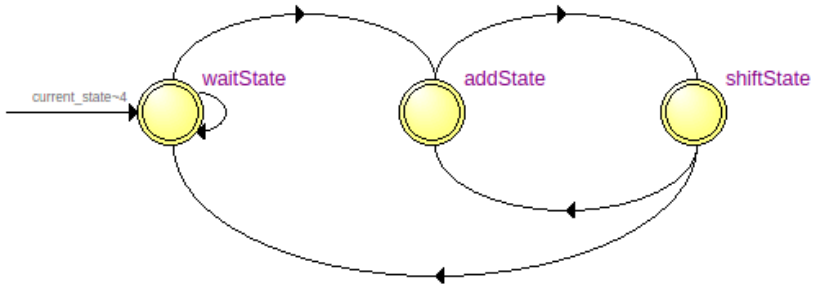

Figure 6: Mealy Finite State Machine

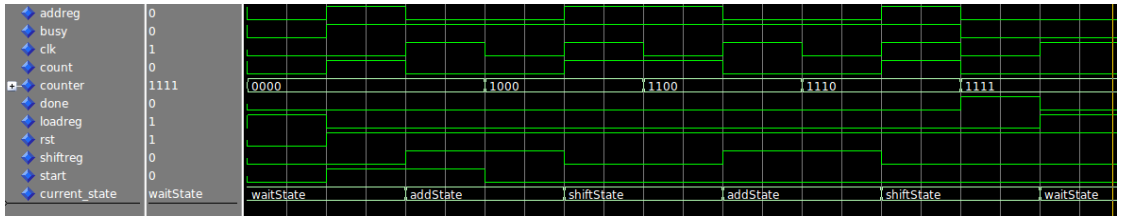

Figure 7: Control Unit Test Bench

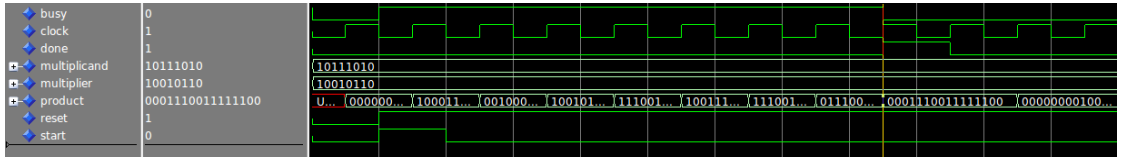

Figure 8: Final Circuit Positive Output Test Bench

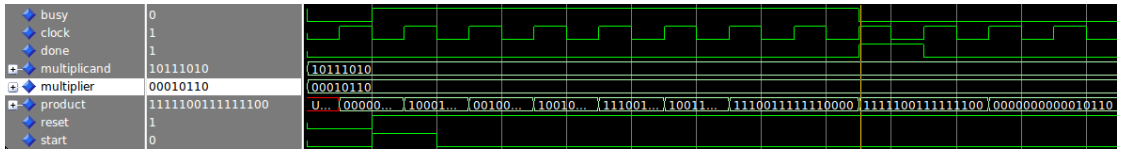

Figure 9: Final Circuit Negative Output Test Bench

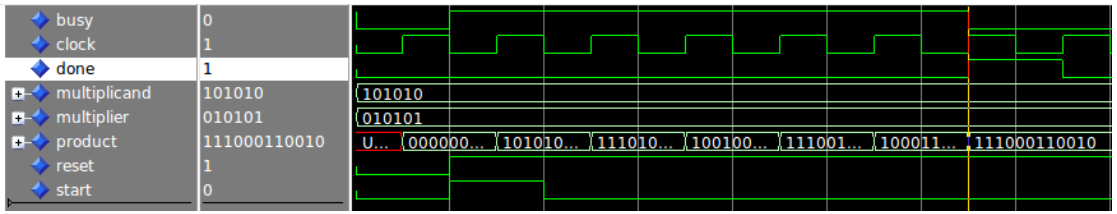

Figure 10: Final Circuit 6-Bit Input Test Bench

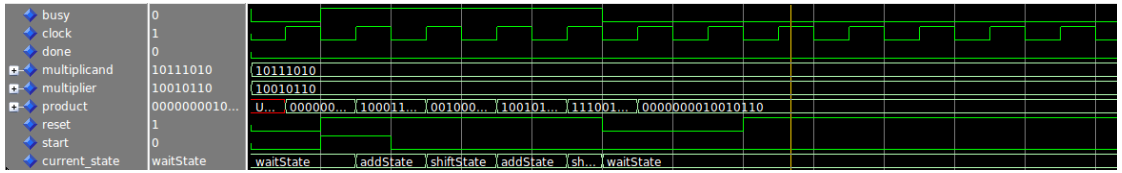

Figure 11: Final Circuit Reset Test Bench

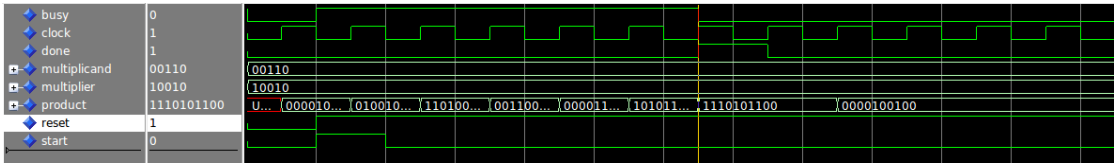

Figure 12: Final Circuit 5-Bit Input Test Bench