Chris Baker
105180929

Project 1: Lab Report

Introduction:

Verilog is a hardware description language that provides a textual format to build a variety of electronic systems including combinational and sequential circuits. The purpose of this project is to allow the students to become familiar with Verilog HDL by reviewing familiar concepts from CS M51A while practicing the development tools provided by Verilog through Xilinx ISE development environment. Our lab zero includes projects on a combinational circuit, four-bit counter, modern 4 bit-counter implementation, and a clock divider. Without the opportunity to work with an FPGA board in class we will speculate and infer on the information that we do receive briefly throughout the report. Generally, an FPGA is a field programmable gate array that can be used to configure a circuit for a specific purpose. The FPGA is made up of logic blocks, interconnects, and I/O blocks which allow the board to implement logic, connect inputs and outputs, and allow for communication with other devices. In order to work with a FPGA board, it is important to know how its design will interact with the physical world which requires proper planning. It is essential to know our goal and the purpose of what we are planning in case we were using the FPGA board and to implement concise well executing programs or circuits.

For our combinational circuit example, we need to construct eight gates intended to perform the NAND, AND, NOR, OR, XOR, XNOR, XOR, Not, Non-inverting buffer operations. These operations will be connected by wires to an 8-to-1 multiplexer that will select the appropriate choice based on a 3-bit value. On the Nexys3 board we would have switches that would allow us to toggle values to set bits used by the gates and other switches to set bits to decide the selector for the mux. The result would be witnessing if the led received a output set to 1 which would turn it on or if it was not on then it would have received a 0. The sequential circuit demonstrates the understanding and ability to implement a counter using traditional methods. We start with designing a 4-bit counter using gates, a clock, a reset, and some feedbacks. The new counter implements a counter using a 4-bit vector and concise code to count each step and assign those values to the register. This example is intended to show the software capabilities, but also to remind us that we should understand the basic underlying principles and hardware before we advantageously or mistakenly use powerful code. The last example we will visit is the clock divider. In this example we would simulate the taking of a clock input such as that from a device or FPGA and the we would divide the opposing frequencies and use a counter to create a resulting 1hz blinking led. This led would blink at the 1hz frequency on the FPGA board if we implemented it during regular CS M152 Lab conditions.

UCF File:

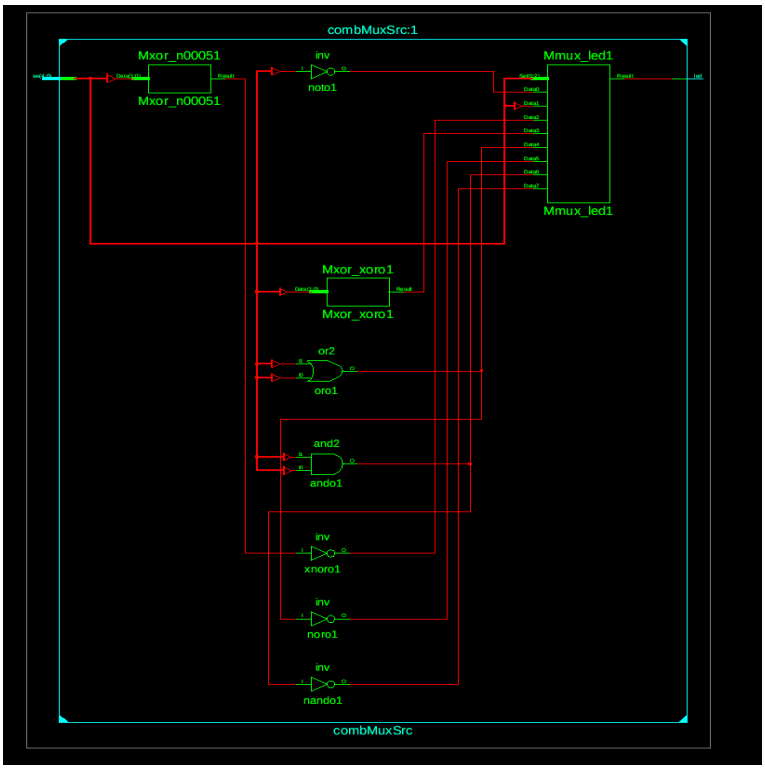
A ucf file stands for User Constraint File which list all the available pin mappings that are available in the FPGA. Xilinx UCF can be applied to various constraints on the design for Xilinx tools such as pin locations, timing, and area. When mapping VHDL to a device you need clocks and pin constraints at the top level. To use the Nexys3 you must plug it in the USB port of PC and start the Adept software, turn on the board, and get it recognized by the pc. In order to set the pins on the board you need to follow a format to map your signal to that location or pin value. For instance, you could write: Net “sw<0>” LOC = 10 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L29N_GCLK2, Sch name = SW0. Notice that we send signal values to certain locations or pins like sw<0> and LOC = 10, #Bank, and pin name.

Chris Baker
105180929

Design Description:

The combinational mux consists of an input vector representing the 5 switches on the FPGA board, an output reg for the led on the FPGA board, and a variety of wires to complete gate operations. The code below shows how assignments were made to connect wires to abstract gates given the logical implementation of how they are intended to work. There are also key words in Verilog for implementing certain gate features, but I opted to stick with typical operations to construct the gates. Vectors for mux inputs and select inputs were also created to represent the wires that would take those appropriate values from the gate and other modular results and forward them to their respective destinations.

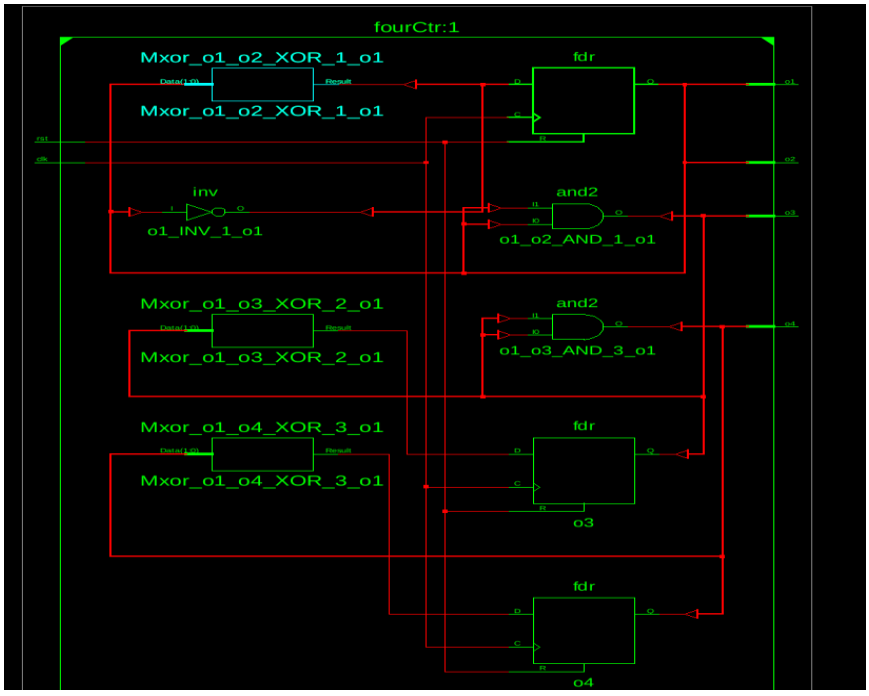
In the schematics for the combinational circuit you can see the clear input and outputs we assigned the main module. From the top view there is the 5-bit vector switches as the input and the resulting output of the module is the led output. Diving deeper into the module we see the switches are sent to various gates except for the non-inverting buffer which heads to the mux directly as an input. Even though the 5-bit vector switch is shown connecting to the gates, only the s0 and s1 bit are being delivered to those gates and the s2, s3, and s4 bits are being delivered exclusively as the select input bits. I took note of a clever move on the Verilog and Xilinx ISE in the creation of this schematic. Particularly, the design decided to take the result of operations like OR and decided to branch off one path to the mux and another to the NOR gate even though my code made it seem like there should be an extra wire or redundant uses of the same values. Thus, like other compilers, there is some efficiency introduced in the execution in building the modular architecture and interactions between the components.



Chris Baker
105180929

The sequential circuit four-bit counter took an input for a clock and a reset as well as four different output registers (o1 to o4). In planning to design this circuit we needed to come up with appropriate designations of wires and registers and assign them with our best interpretation of recreating the lab manual diagram. Since we were designing a counter based on 4 different registers I decided to keep the output values separate and reset their value to 0 if the reset value was encounter and if not then we would perform the combinational logic using non-blocking value setting to the registers. For this flip flop like counter we know that if the clock encounters a positive edge and the reset value is high, then all the output registers should be set to 0. We did not initialize them originally, so we know they start as dc values until this first occurrence of both a positive edge and zero occurring to force the count back to zero. If the positive edge of the clock occurs, but the reset value is not set, then the counter begins to count using sequential circuit logic. This is done by implementing NOT, AND, XOR, and feedbacks with the register values. Effectively, the o values represent the four bits that make up the counter which ranges from zero to fifteen. In order to execute this logic effectively non-blocking statements must be used so the values of o change in parallel since they are dependent on each other's values.

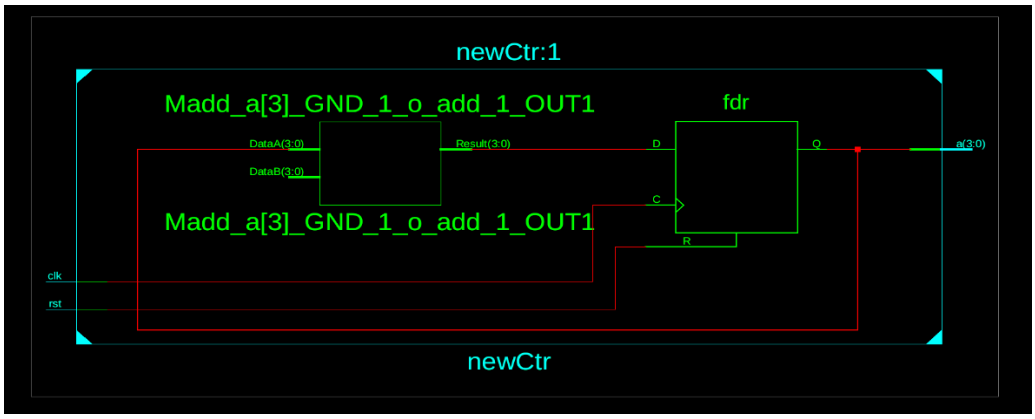
The four-bit counter schematic shows a somewhat similar design to that seen in our diagram on the lab manual. The main top-level module takes an input of reset and clock and has the 4 corresponding o outputs to determine the total count value. Within the module are several registers that appear like the D register. The clock signal branches into each individual clock inputs of the submodules and the reset signal is mapped to the submodules reset values as well. Inverted gates, XORS, and AND gates are used to implement the rest of the logic to create the outputs which are forward to the respective o values.



The new counter was based on the lab manual as directed by our TA. Using this example as a framework we can utilize the benefits of programming and electronic design systems that have been improving over time. The input to the modern counter takes a reset and clock value like the previous

traditional method. This time we assign a four-bit vector as an output register. In this case the values increment one at a time to count in a simple manner. If the value were to try and increment to sixteen then it would toss the extra bit and keep to only four-bit values of zero. As we expect this is clean and efficient. Initially the values here are net set until a positive edge of the clock and the reset value are both detected. When this happens simultaneously the four-bit vector is set to zero. For any following positive edge clocks there is a test if reset happens or not. If reset occurs, then it will force values to zero again. If not, then it will count as intended from zero to fifteen.

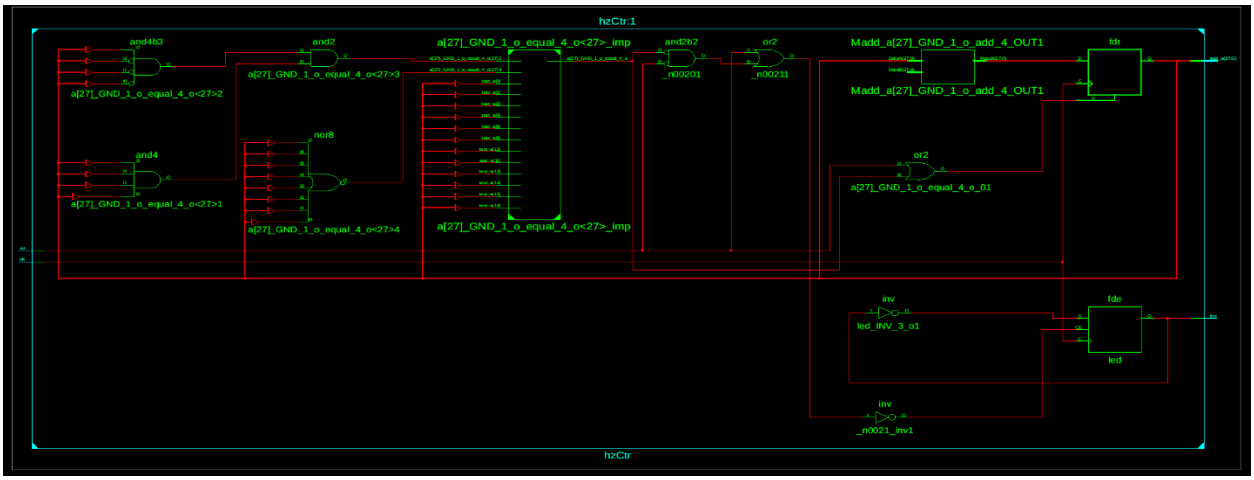
The modern counter has a much cleaner schematic where the top level still takes an input of clock and reset, but the output is delivered as a four-bit vector. Within the inner workings of the module is like a D flip flop. This flip flop performs a feed back sending its output to a submodule performing a add out and returning the result back into the input D of the flip flop.



Our final design description pertains to the clock divider. This design is intended to take a ten thousand kilohertz clock signal and translate into a flashing led with a one hertz frequency. In order to design this, we knew there would involve a counter and we would need to designate the input clock in our testbench while maneuvering logic and values to create the output clock that would yield our one hertz flashing led. Knowing that a counter would be implemented I set up a module to hold various ports including reset, clock, led, and test. I was not sure how many bits I needed at first, so I allowed the counter to be a high bit value in case I needed a larger number. For the logic I decided that a at positive clock edges I would check for a reset. If I encountered the reset here, I would change the counter value to zero. If not, I would check if the counter was approximately half the needed value to create our one hertz output clock from the ten kilohertz input clock. If we detected our target value, then we flipped the led and reset the counter. If neither of those cases occurred, then we incremented the counter as normal. For this example, I looked up SI conversions and did arithmetic to try and see possible combinations to achieve the divider. I worked with many test cases and number switching to come to my desired result and a nice test simulation. If we had an FPGA board, we may have been able to implement interconnects between the board and Verilog to test differences of the clocks and try using those to divide and test the led output.

The schematic for the clock divider takes the reset and clock signals as input and outputs the large bit vector and led signal. Within the clock divider are numerous gates including AND, NOR, OR, INVERTER, and a mux and add out. The mux seems to implement part of the dividing logic by deciding which values are forwarded to some of the other sub modules to go into our flip flop sub module. The

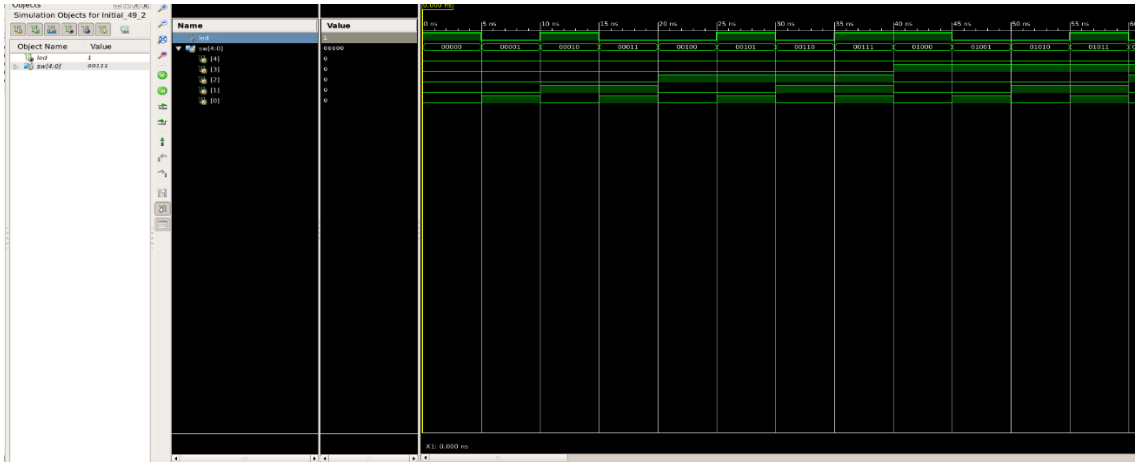
inputs to the mux consist of several of the gates mentioned above and make up the selector as well. These parts make sense because my code implements the counter to count every clock edge, but perform different operations depending on the value of the register at that moment. That value decides if we will start counting all over again, if we flip the led, or if we proceed counting.



Simulation Efforts:

When testing the combinational circuit, I encountered entry level issues working with Verilog and Xilinx ISE. I originally assigned incorrect variables like reg or wire and was received values like don't care (x) for testing values. I was able to fix this by properly assigning wires and using reg variables on the LHS in always blocks. The rest of my issues occurred during testing where I may have made an error trying to hand write twenty test cases. I repetitively added time unit statements as well as various formats for declaring numbers in Verilog. While experimenting with values I eventually started getting errors and could not see the outputs anymore. I decided to ditch the multiple test cases and implement a counting test case to test every case value. As a verification I tested a few cases to doublecheck the accuracy. The top 3 msb bits of sw determine the select for the mux and the lower 2 bits of sw determine if either of the two inputs for the gates are set. If select is 000 then we use the NOT gate, given 0 as input to the NOT gate we return 1 to the led. At 5ns we change the input of the NOT gate to be 1 and we see the result is 0 effectively turning off the led. At 40ns the XNOR gate is selected and the inputs passed to the gates are 0 and 1 which results in 1. The test completes all cases by 160ns, but we run it till 200ns and notice it begins counting back up from 0 at 165ns.

Chris Baker
105180929

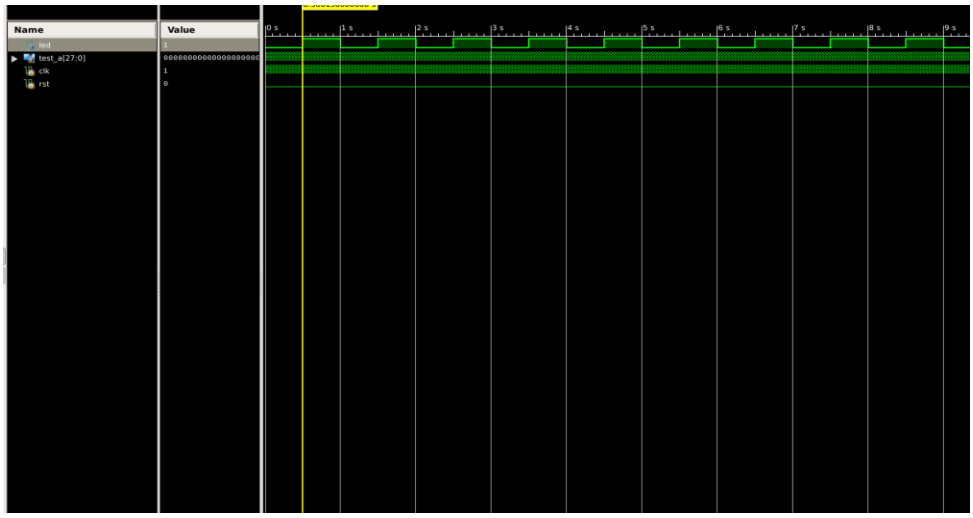


The sequential circuit for bit counter was troublesome because I kept having an error from not making the register from my parent module a wire in the test bench. Further, I was missing explicit declarations under the UUT (unit under test) which did not allow me to properly associate values as I intended. Even after resolving these issues I was mystified by the red line that started at 0ns in my waveform. However, this one was not an error and caused me to make changes that resulted in assignment errors until I went back to my correct code. The red line is there because I did not initialize the values and the values will become initialized when the clock and reset signals are both high (see 25ns). I proceeded to check the values for my counter, and we see that at 55ns it is set to 0001. Following this the second value is set yielding 0010. The counter accomplishes its counting purpose and we later test that if we set a clock signal and reset signal to be high at any point during the counting then we will yield a 0 result. This can even be noted at 25ns where the values are initialized to 0000 but will not start counting since they are held by the reset occurring with the clock.

Our goal with the new counter was to compare the waveform results and test a different method of implementation after understanding the underlying hardware. Once I figured out my errors in the previous counter, I was able to fix similar errors of output assignment in this part of the project. Notice that we have the same don't care value in the beginning like the traditional counter. Further when we hold the reset and clock signals both high the values are stuck at 0000 longer then the following counting that occurs. Once we obtain the value of 0001, we count at equal timer intervals with

the clock just like our earlier counter. Lastly If we send the reset signal again, it will not be noticed until the positive edge of the clock occurs with them both on. This shows our resulting 0000 that lasts for the rest of the photo of the schematic. These results were all similar and prove the accuracy of both counters.

For the last waveform we verify that we were able to create a 1hz frequency led. This is seen with the 1s equally spaced between each time the led turns off. In testing this I did SI conversions and arithmetic on paper to try and achieve target values that could result in the right frequency dividing and counting logic. I had trouble achieving these values for a while, but my error here was having little understanding of clock dividers and trying to learn how they work. After checking numerous examples, I saw different implementations, but was stuck on getting the 10kHz signal to come out with 1Hz. I started by determining the time period resulting from the clock in. Then I determined how I would set my clock timer to turn it on and off. Then I ran various numbers into my counter to see a trend of the frequency getting closer to 1Hz. I kept testing trial and error and achieved an output of 1Hz. I did have to fix some assignments here but learned how to handle that with test values and from my understanding of errors from earlier problems with the counters.



Chris Baker
105180929

Conclusion:

Project 1 encompassed a variety of topics introducing Verilog and Xilinx ISE while reviewing somewhat familiar topics ranging from combinational circuits to clock dividers. The combinational circuit consisted mainly of gates, wires, a mux, and a register. The gates provided the needed operations to be the selected inputs of the mux and the wires connected most of the data between sub modules. The led was a register so it could store the value in effect the led would be able to turn on or shut off which we would be able to see if we ported it onto a FPGA board. Understanding the diagram from the lab manual was crucial and I followed that as a guide to where I intended my mux inputs to select the values across its bit vector. The Sequential Circuit required proper use of non-blocking statements and correct assignments for wires and the placement of registers in the various code blocks. The design here was largely based on the diagram provided in the lab manual from which I mapped the resulting values before and after each gate to come up with the logic assignments determining my output values. Since we were designing a counter based on 4 different registers I decided to keep the output values separate and reset their value to 0 if the reset value was encounter and if not then we would perform the combinational logic using non-blocking value setting to the registers. The new counter was a different implementation that resulted in identical results. This code was taken from the lab manual and I filled in a test bench to test check these results. The new counter was designed with using a counter incrementing values every clock without coding gate logic directly. The clock divider was another counter-based design which involved resetting, but at a determined value that was dependent on the relation be the input clock.

Difficulties encountered include several errors on setting values with variables. I did not always declare values or instantiate them which led to high impedance or don't care being set. Other times I received error codes stating a wrong port was used or a wrong type was declared from the source file to the test bench. I was able to read up documentation by following the error code links on Xilinx and sometimes noticed incorrect assignments. When I did not notice them, I defaulted to trial and error testing register in place of wire for some variables. From this I would deduce if I was moving further or closer to my intended result one step at a time. Final errors included misuse of number formatting or values used especially in the clock divider. I had to double check the correct format for decimals and binary digits to enter these values correctly.

Future recommendations for the lab include clarifying the clock divider problem. I know there was an adjustment to the writing since we are not doing this project in class, but I was not sure if the 1Hz was intended since it was written as '1 -Hz' and I was comparing it to '4-kHz'. The space in-between made me think maybe there was an error in typing. Also, I had to read that example a few times to understand the purpose unlike the other examples which seemed a bit more straightforward. For pre-lab prep I would recommend some more time be used for RTL Schematic process and understanding the connection between variables in the testbench compared to the source file. I spent copious time trying to iron out issues in solving these errors and trying to make schematics (one by one in new projects), so that would be appreciated for future classes.