ECE 3270 : LAB 1

FIBONACCI CHECK AND SEQUENCE GENERATOR

February 28, 2022

Aaron Bruner
Clemson University
Department of Electrical and Computer Engineering
ajbrune@clemson.edu

ABSTRACT

For this lab we were tasked with developing various different circuit elements that would perform as expected for us. The first component was a 2-to-1 multiplexer with a select input. The next component verified Fibonacci numbers. This component takes in a four-bit input and outputs a logic one or zero for true or false. The next component determines the next Fibonacci number based on a four-bit input. The output is five-bits instead of four since we need to consider the value 21 being the next Fibonacci value after 13. The final portion is to combine all of the previous components to develop a system that can determine either the current or next Fibonacci number and display it on the 7-segment display. In addition to this, an LED will light up if the four-bit input is a Fibonacci number.

1 Introduction

The purpose of this exercise is to learn how to connect simple input and output devices to an FPGA chip and implement a circuit that uses these devices. We will use the switches (SW_{5-0}) on the DE10-Lite board as inputs to the circuit. We will use light emitting diodes ($LEDR_8$) and 7-segment display ($HEX0$) as output devices.

The DE10-Lite board provides 10 toggle switches, called SW_{9-0}, that can be used as inputs to a circuit, and 10 red lights, called $LEDR_{9-0}$, that can be used to display output values. Since there are 10 switches and lights, it is convenient to represent them as arrays(STD_LOGIC_VECTOR) in VHDL code.

The DE10-Lite board has hardwired connections between its FPGA chip and the switches and lights. To use SW_{9-0} and $LEDR_{9-0}$ it is necessary to include the correct Quartus project pin assignments which are provided. For example, on the DE10-Lite board, SW_0 is connected to the FPGA pin $C10$ and $LEDR_0$ is connected to pin $A8$. A good way to make the required pin assignments is to import into the Quartus software the file called DE10_LITE.qsf for the DE10-Lite board, which was provided.

It is important to realize that the pin assignments in the .qsf file are useful only if the pin names given in the file are exactly the same as the port names used in your VHDL entity. The file uses the names $SW[0]$ ::: $SW[9]$ and $LEDR[0]$ ::: $LEDR[9]$ for the switches and lights (notes that the Quartus software uses [] square brackets for array elements, while the VHDL syntax uses () round brackets). The 7-segment displays behave differently, with each labled as $HEX0 - HEX5$. Each display has 7 segments, indexed by $HEX0[0]$ ::: $HEX0[6]$.

2 Design

This section discusses how the components were developed and then eventually how they were combined to create the overall lab.

2.1 Designing a 4-Bit Multiplexer

The first component needing to be created is a multiplexer. The propose of our multiplexer is to take in multiple inputs and produce one and only one output. This is done using a select line which we denote with s. This is a somewhat simple thing to implement into VHDL considering we have a simple example on slide 48 of lecture two [2]. In addition, we were provided with an image that represents the circuit on page two of the lab assignment [1]. Converting the circuit to VHDL code is easy when you have such a simple circuit.

The multiplexer is a somewhat simple implementation. We first need to think about our variables that are in the multiplexer. Based on the images in the lecture slides we can think about inputs, select line, and the output. For our lab, we need to create a 4-1 multiplexer. This means we will take in a 4-Bit binary value and then output a 1-Bit value. For us, we have two 4-Bit input lines, x and y respectively. Next, we will have a select line which is denoted with the letter s. Finally, the output is mapped to the m line. The architecture is a replica of the code on lecture slide 48 [2] but with only two inputs instead of four.

Based on the lab manual we want to output x when the select line is logic low and output y when the select line is logic high. In our code, we set the output to x when the select is 0 and y when select is 1; for any other value we ignore it. Considering we only are allowed a logic high or low we are not considered with other input values. We can validate the results by simulating our code, this is shown in figure 3.

2.2 Design a 4-Bit Fibonacci Detector

The next component that is needing to be developed is the Fibonacci detector. For this component we will use the same logic from our multiplexer. The input is going to be a 4-bit binary number and a 1-Bit output. The idea is that we will input some binary number and then compare that to predetermined Fibonacci numbers that we are interested in. For this lab, we are only interested in the Fibonacci numbers 0, 1, 2, 3, 5, 8, 13, and 21. Thus, we only need to check for those exact cases.

The architecture begins with a select statement for our input line. We will compare that to the binary versions of the Fibonacci numbers we are interested in. If the input is one of those predetermined Fibonacci numbers we set the output to a logic high. If it is not then we need to set it to 0. Thus, for the inputs 4, 6, 7, 9, 10, 11, 12, 14, and 15 we get a logic low. We can verify that this is working by referencing figure 4.

2.3 Design a Component to Find Next Fibonacci Number

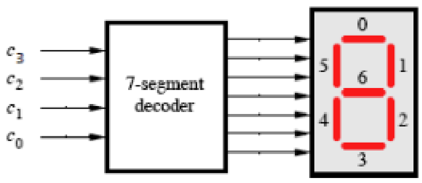
The next component needed will determine the next valid Fibonacci number. Similar to our previous component we are going to input a 4-Bit binary value and then output a 5-Bit binary value. Our 4-Bit input could be any value from 0 to 15. For the output we are only going to see valid Fibonacci numbers that we are concerned with. The only values we are concerned with are 0, 1, 2, 3, 5, 8, 13, and 21. For any other value we are going to output 11111 which is a requirement from the lab manual [1].

We verify our results by inputting all values from 0000 up to 1111 which is the full range of values we could deal with. As shown in figure 5 we can see the valid Fibonacci numbers being generated to begin with. Then, as we begin to input invalid numbers we see the expected result of 11111.

2.4 Design a 7-Segment Display Component

The next section is designing a component that can display a value to our 7-Segment display. The 7-Segment display can display numerical values ranging from 0 to 9 and alphabetical letters from A to F. The only values we are interested in are 0 to 9 and the value E. We will use E to know that we have an invalid Fibonacci number. For any valid Fibonacci number, other than 13, we will display it on the 7-Segment display.

The lab manual demonstrates all of the valid outputs of the 7-Segment display based on our 4-Bit input [1]. This can be seen in figure 1 shown below. Thus, all we are tasked with doing is managing our 4-Bit input and then outputting the correct 7-Bit value for later use. So, as with all other components, we duplicate our code and change variables and numbers. This time, we are inputting a 4-Bit binary number and then going to output a 7-Bit binary number. The only values of interest are 0, 1, 2, 3, 5, and 8 since that is all we can display on a single 7-Segment display. For any other value we are going to output our E.



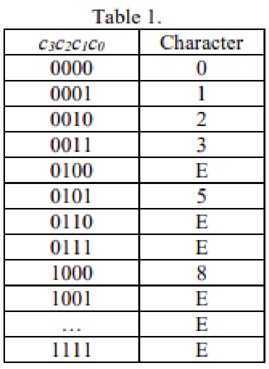


Table 1.

$c_3c_2c_1c_0$	Character
0000	0
0001	1
0010	2
0011	3
0100	E
0101	5
0110	E
0111	E
1000	8
1001	E
…	E
1111	E

Figure 1: Lab Manual representation of the 7-Segment display values.

2.5 Combining all Components

The final section for this lab is to combine all of the previous components of this lab into one. The purpose of this is to demonstrate knowledge of combining components from separate files into one circuit. Thus, we begin as normal by declaring our input and output pins. For this part we are going to be inputting a select signal for our multiplexer from part one. The next input is the data which will be our 4-Bit Fibonacci value. The next is an output of 7-Bits which will be to display on our 7-Segment display. Finally, we will have a 1-Bit output for the LED to determine if we do have a valid Fibonacci number in data.

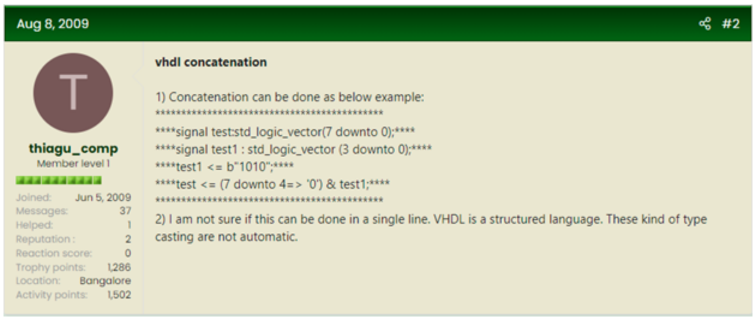
Next, we will describe our architecture. To begin the architecture, we need to add signals which can be though of as actual wires. Or in typical programming languages like C these are variables that can transfer information to different components. We will need three different signals for our circuit to function. The first is a 5-Bit binary number which will carry the next Fibonacci number to a multiplexer. Another signal is named the longData which is our normal data but with an extra zero at the beginning. This is explained further later. Finally, we need another 5-Bit binary value which will be the value that we want to display onto the 7-Segment display.

To begin, we will add all of our components starting with the multiplexer. This is how to prepare our circuit for when we start to reference specific components. All we do is list the components file name and then type out the PORT which is a list of the components inputs and outputs.

Once this is complete we can start describing how our components will work together. It's best to think about what we will be first presented with. First, the board will have a binary input of all zeros across bits 3 to 0. In addition, the select line will be set to 0 which means we will display the current Fibonacci number. Thus, we should start by checking if the input is a Fibonacci number; we do this by first giving our Fibonacci checker the data and the output isFib which will go to our LED. If it input is a valid Fibonacci number then the LED will light up. Either way we continue to the next component of part five.

The next step is to determine the next Fibonacci number. We do this by including our part three which takes in a 4-Bit value and outputs a 5-Bit value with the next Fibonacci number. The reason it is 5-Bits and not 4 is because we are also considering 21 as the next Fibonacci number to 13. Once we have this information we can transfer the output to one of our signals so that we can use it later in the circuit.

The next component is to determine whether we are going to output the current Fibonacci number or the next Fibonacci number. We load the current number on the x line of our multiplexer and the next value on the y line. We can select either value by a switch (SW_4). There is only one issue here. There is only one issue; in the last step we provided a 4-Bit value and were given the next Fibonacci number which is a 5-Bit value. We need to have the same size values for both our data and the next Fibonacci number. We put this corrected value in the signal longData. After some research it was found out that you can apply concatenation to a value and practically add another zero to the beginning. This information can be found in the image shown in figure 2.



Aug 8, 2009 #2

thiagu_comp
Member level 1
Joined: Jun 5, 2009
Messages: 37
Helped: 1
Reputation : 2
Reaction score: 0
Trophy points: 1,286
Location: Bangalore
Activity points: 1,502

vhdl concatenation

1) Concatenation can be done as below example:

```
*******************************************
****signal test:std_logic_vector(7 downto 0);****
****signal test1 : std_logic_vector (3 downto 0);****
****test1 <= b"1010";****
****test <= (7 downto 4=> '0') & test1;****
*******************************************
```

2) I am not sure if this can be done in a single line. VHDL is a structured language. These kind of type casting are not automatic.

Figure 2: Web article describing concatenation. https://www.edaboard.com/threads/vhdl-concatenation-and-integer-to-single-bit-question.155878/

Once this issue is resolved we can continue developing our circuit. With the two inputs to our multiplexer being the same size we can output the desired value using the select line. After this, all that is left is displaying our output to a 7-Segment display. We transfer the output of our multiplexer into our signal MUXoutToDisp which is then put into the fourth part. We input a 5-Bit value and then output a 7-Bit value which will be sent to the 7-Segment display.

All that is left is to verify that this is functional by using a test bench. We can see that based on the input and the select line we get the next Fibonacci number or the current sent to the 7-Segment display. In addition, the LED is either on or off depending on the input values from the data. We can see these results demonstrated in figure 7.

3 Results

After completing all of the simulations and trouble shooting in Quartus we are ready to implement our code onto our board. Once we compile the final code we can open our programmer and then put the compiled code on the DE10-Lite. After we start the programming we will see all of the lights freeze for a second then all turn off. Next, the 7-Segment display will display zero if all of the switches are set to 0. We decided to use LED number 9 to display whether or not it was a valid Fibonacci number or not. In addition, we are using switch 4 for the Multiplexer to change between the current Fibonacci number and the next. Lastly, our data input is going to be switch 3 through 0 representing a 4-Bit data line.

After we have it programmed on the board we can switch the switches and iterate through the values to ensure functionality and even test having switches in between to see what happens. To no surprise, nothing interesting happened. The code we developed handles incorrect input rather well.

4 Conclusion

To conclude, this lab taught us many things about Quartus, ModelSim and our DE10-Lite. For starters, transferring code from the Linux machine to a Windows machine is a very big mistake. I had to recreate my project folder and copy and paste code over to return functionality. Next, the DE10-Lite does not retain the information after programming. In other words, after you power cycle the device it completely forgets what you programmed to it. It's almost like it's only simulating our code, not actually programming it locally. Finally, there are many useful shortcuts in Modelsim that help us represent our graphs. Shortcuts like F resize the graph to the region where data is being presented. Another useful shortcut is H which changes the name of the signals to make it easier to read.

One more thing to mention, throughout the development we had to constantly modify the size of our signals to be consistent with new components that we were creating. This is best represented in the code and is documented line by line for ease of reading.

References

[1] B. Green. Lab 1: Fibonacci check and sequence generator. PDF, Clemson, 2022.

[2] Y. Lao. L02.vhdl. PDF, Clemson, 2022.

5 Appendix

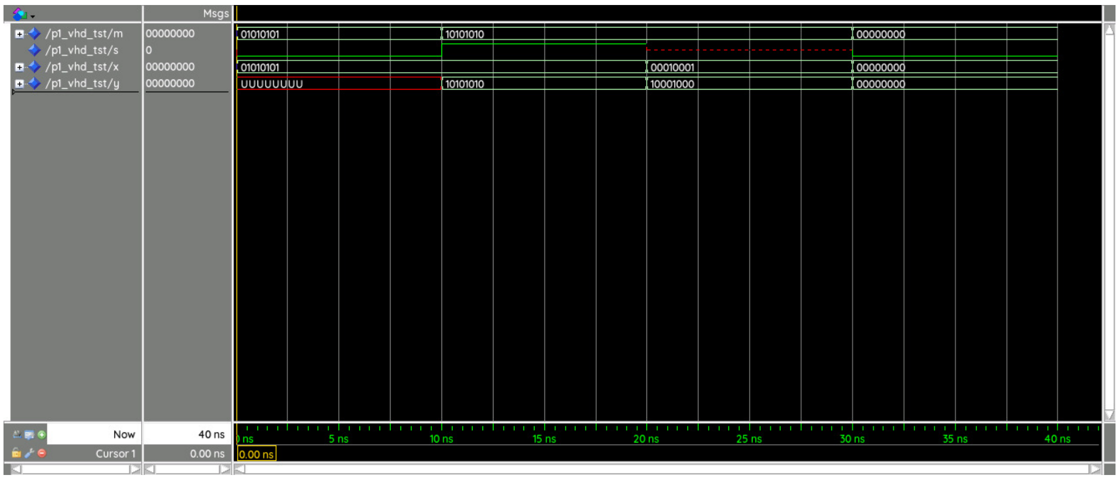


Figure 3: This is the 4-Bit Multiplexer simulation. This demonstrates how the multiplexer functions.There are two 4-Bit input lines, x and y. These are the data lines that we use to transfer our input to the output m. Based on our selection of s we can choose between x and y. When s is 0 we see x on the output line and when s is 1 we see y on the output. From 20ns to 30ns we tested an invalid input and saw no difference in the output.

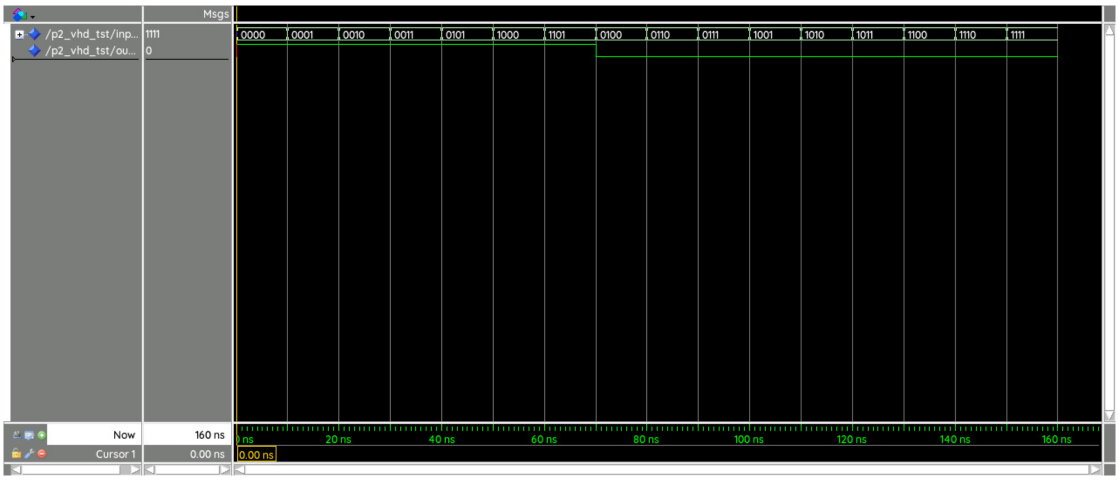


Figure 4: This is the 4-Bit Fibonacci number verification component. The input line will have a 4-Bit value and will compare it to the values we predetermined to be valid Fibonacci numbers. If the value is a Fibonacci number we output 1. If not, then we output 0. For this figure, we listed all valid Fibonacci numbers first, then all invalid next. We can see that 0, 1, 2, 3, 5, 8, and 13 are all valid Fibonacci numbers.

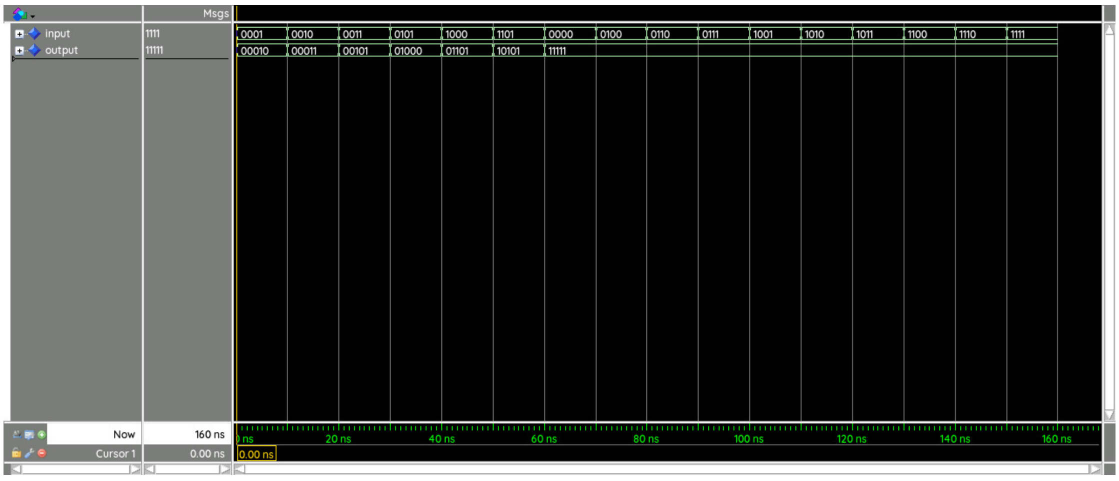


Figure 5: This portion determines the next Fibonacci number based on the input number. The input line is a 4-Bit number and the output is 5-Bits. This way we can support having 21 as a valid next Fibonacci number.

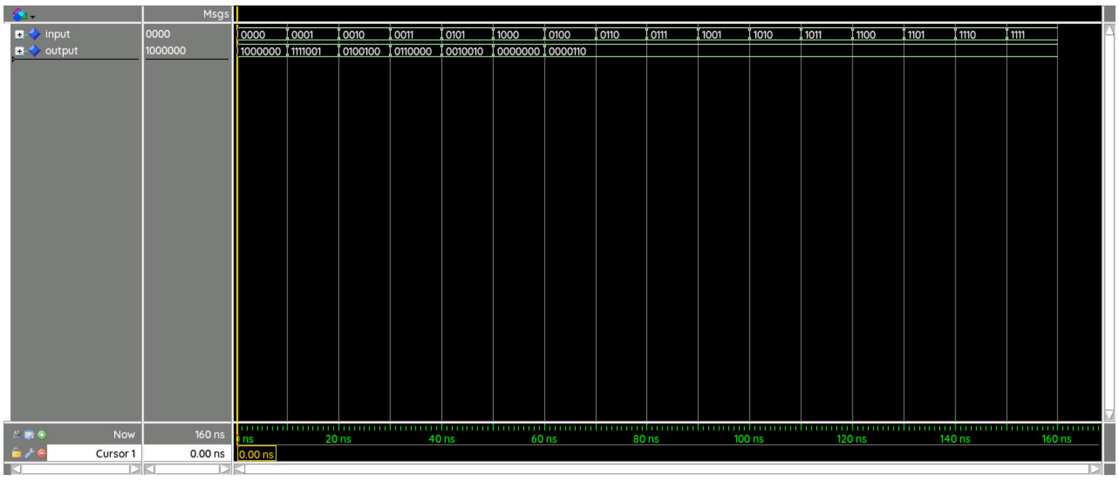


Figure 6: The image demonstrates what to output on a 7-Segment display for each input value. We listed the valid inputs first and then all invalid inputs last. For all values with output 0000110 we will see an 'E' on the 7-Segment Display.

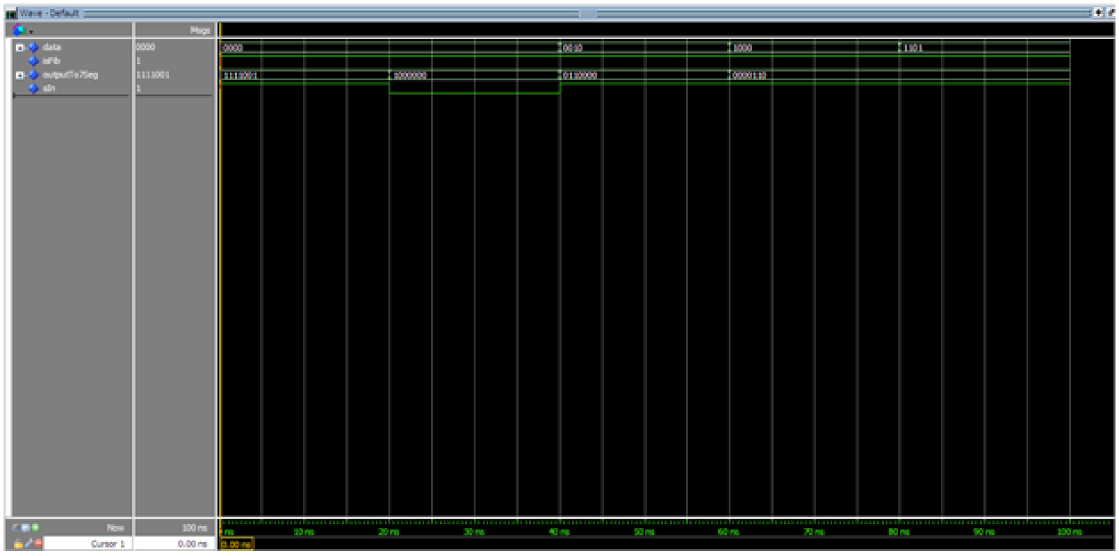


Figure 7: The image shows a simulation of the combination of our entire circuit. When we input select 1 we are going to be finding the next Fibonacci number. Considering our input data is 0 we are expecting 1 on the output which is 1111001 for the 7-Segment display. The zeros indicate the light being on. For the rest of the combinations we can use the same logic to verify functionality.