LAB 3 –Verilog for Combinatorial Circuits

Goals

- Learn how to design combinatorial circuits using Verilog.
- Design a simple circuit that takes a 4-bit binary number and drives the 7-segment display so that it is displayed using hexadecimal format.

To Do

- Design a circuit that accepts a 4-bit binary number and generates the 7 control signals that drive the 7-segmetnt display.
- To demonstrate the circuit use the circuit from Lab2 and attach the output to the 7-segment display.
- Follow the instructions. Paragraphs that have a gray background like the current paragraph denote descriptions that require you to do something.
- To complete the lab you have to show your work to an assistant during any lab session, there is nothing to hand in. The required tasks are marked on the report sheet at the end of this document.
  All other tasks are optional but highly recommended. You can ask the assistants for feedback on the optional tasks.

Introduction

In the previous exercise we have used LED lights to display the result of our adder. While it is possible to read binary coded numbers, it would be interesting to use the large displays on your board that can display numbers. Such displays are called 7-segment displays from the simple fact that they consist of 7 separate LEDs in a single package (see Figure 1). Each of the seven segments is labeled a through g. By controlling which of the LEDs light up different characters can be displayed. In this lab we implement the coding shown in Figure 2, which acts as a hexadecimal display. If we want to display 0, we should make sure that all of the LEDs light up except ‘g’.

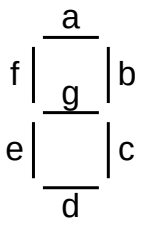

Figure 1. 7-segment display

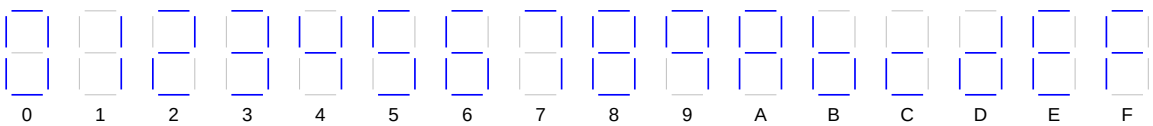

Figure 2. 7-segment display function

Each one of the 7-segment connections is directly connected to the FPGA, just like the LEDs in

the previous exercise. What we have to do is to convert a 4-bit binary input to drive the correct 7-bit signals. One thing to note is that, unlike the LEDs which light up when a logic-1 is applied, the seven segment displays light up when a logic-0 is applied. Such outputs are called active-low, and have to do with the way the circuit has been built.

You will notice that there are a total of four 7-segment displays on the FPGA board. To save on the number of connections, all four 7-segment displays share the same inputs. By default all four of them will show exactly the same number. Each of the displays has an additional activation input. When 4 digits are used, a sequential circuit starts from the first digit, activates the first segment and drives the number to be displayed, then (in the second cycle), the first digit is deactivated, and the second digit is activated, and the corresponding digit is written out and so on. Since the human eye can only notice changes which are slower than approximately 20ms, if you do this fast enough (more than 50 times per second) you will see a stable display. This is the same trick used in movies and television sets. In this exercise, we do not implement this trick (we have not yet learned how to do sequential designs) and so we must accept that we see the same number four times. However if you are bothered by the repeated output, the three extra 7-segment displays can be turned off.

We can reuse our last circuit and place the 7-segment display at the output. The problem is that the output of the adder has 5-bits, which is more than what we have for the 7-segment display. We can simply continue to use a LED from the board for the 5^{th} output.

Circuit Design

As a first step, fill in the truth table in your report for the binary to 7-segment decoding. Note that the segment will light up when the corresponding output is logic-0.

The truth table you have filled defines a circuit with 7 outputs. If you wanted, you could write the Boolean equations for all outputs separately, try to find out a (somewhat) optimized version of the equations, see if you can share portions of the gates among the different outputs (reducing the number of gates) and then draw the schematic for the resulting Boolean equations.

This is not very difficult and can be done with some effort. However, since we have learned that we can also use Verilog to describe hardware, we can do this faster.

Before we continue, remember the bus argument from the previous exercise. In this design, we have 7 outputs that we have named A to G. Instead of making 7 separate connections we can bunch all of them together in one bus. Assume that you call this bus D(6:0). The important thing is to decide on how the bus is connected to individual outputs. Is A connected to D(0) or to D(6) (or to another one)? You are free to choose whether or not to use a bus, or what kind of assignment you want to make to the buses. We assume that you call the output D(6:0) and A is D(0), B is D(1), C is D(2) etc. For the inputs, assume that we have a 4-bit bus called S(3:0).

Continue working on your old Lab2 project by starting Vivado. Inside Flow Navigator, under "Project Manager", click on "Add Sources". Select "Add or create design sources", and click next. Click "Create File". A new dialog box as follows will pop up. After you choose a name for the module (e.g. 'Decoder') clicking on the 'Next' button will open a new window where you can easily add the connections. You can type in the name, select the direction, and if you are using buses, you can enter the range. Once you are finished, the user area will turn into a text editor where you can write the Verilog code for your decoder.

Now you are free to describe the Verilog code for the decoder as you see fit. You can try different styles to see which one suits you well. It is important to realize that the length of the code is not proportional to the complexity of the hardware. Try to write well-structured code using simple statements and document them well.

Note that in case you have a procedural assignment to a wire (inside an 'always' block) you should make sure that it is declared as 'reg'. This is also the case for outputs that are defined in the module declaration. In general all signals on the left hand side of <= or = in an 'always' statement must be declared as 'reg'. See paragraph 4.4.1 of the book if you still have doubts about it.

There is not one correct solution to this problem: there are many variations. Towards the end of the exercise we see our circuit in action and if it behaves correctly.

In case there is any syntax error in your code, you can see a red underline along with a red marker on the scroll bar. Move your mouse to the scroll bar and you can see the cause of the error. Additionally you can go to Window → Messages which also shows all the errors (including the warnings) in your code. You may have to save the file before the messages are updated.

To visualize the schematic, go to Flow -> Elaborated Design, and you can view the schematic of the code. You can ignore the warning dialog.

Adding the decoder

Our goal is to include the decoder between the output of the 4-bit adder that we have previously designed and the outputs of the whole circuit. There are two choices that we have. We either modify the existing adder circuit adding the decoder to the end, or we keep the adder as it is, and add a new source file where we instantiate both designs. Once again, there is not really a 'correct' solution; it is more of a personal choice. We explain the solution where we add a new hierarchy, but if you believe that you would rather do it the other way, feel free to try it out.

Now we need to create a new source file where we instantiate the adder and the decoder that we have just designed. In this module we will still have the two inputs A(3:0) and B(3:0) that go directly to the adder. The output of the adder is a bus that is 5 bits wide. 4 of these S(3:0) will go to the decoder, and the most significant bit S(4) should connect to a separate LED. An example schematic is given in figure 3.

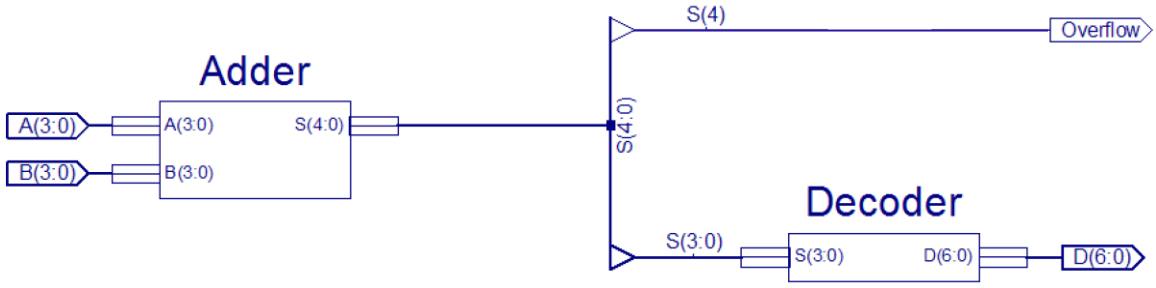

Figure 3. A possible top-level schematic

Using any approach that you deem appropriate, connect the decoder between the adder and the output. Check your design by looking at the schematic, and make sure that you do not have any errors.

We also need to modify the constraints file that tells us where to connect the outputs (the xdc file). We no longer have the output S(4:0), in its place we have the Overflow output and D(6:0).

Go to project manager, select Constraints and then select the .xdc file. Make necessary changes to enable the pins required for the 7 segment decoder.

The connections for the 7-segment display are the following:

```
set_property                          PACKAGE_PIN W7 [get_ports {D[0]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[0]}]
 set_property PACKAGE_PIN W6 [get_ports {D[1]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[1]}]
 set_property PACKAGE_PIN U8 [get_ports {D[2]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[2]}]
 set_property PACKAGE_PIN V8 [get_ports {D[3]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[3]}]
 set_property PACKAGE_PIN U5 [get_ports {D[4]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[4]}]
 set_property PACKAGE_PIN V5 [get_ports {D[5]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[5]}]
 set_property PACKAGE_PIN U7 [get_ports {D[6]}]
     set_property IOSTANDARD LVCMOS33 [get_ports {D[6]}]
```

Now we just have to generate the programming file of the entire project and download it to the FPGA. Then we can finally check the result using the 7-segment displays.

Using Lab2 as a reference, generate the programming file and program the FPGA. Show the working circuit to an assistant.

Optional:

If you don't like seeing the displays showing the same number four times you may want to turn off the three extra 7-segment displays. They can be disabled by connecting their activation input to logic-1 (note that the activation input is active low). In order to achieve this you can add a new 4-bit output an(3:0) to the source file and assign the value 1 to an(1), an(2) and an(3) and 0 to an(0).

At this point the constraints file should be updated in order to map these new output signals to the activation pins on the board.

Do this by adding the following lines to your constraints file:

```
  set_property PACKAGE_PIN U2 [get_ports {an[0]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
 set_property PACKAGE_PIN U4 [get_ports {an[1]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
 set_property PACKAGE_PIN V4 [get_ports {an[2]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
 set_property PACKAGE_PIN W4 [get_ports {an[3]}]
                set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]
```

Of course you can name the outputs as you prefer, anyway just remember to be consistent with your choice when editing the constraints file. After this little tweak, generate the programming file again and reprogram the board. You should now see only the right-most 7-segment display

on.

Last Words

In this exercise we deliberately left some options (how to add the display decoder) up to you. There are frequently many ways a specific task can be completed, and it is not immediately clear which of the alternatives is the best choice. More often than not, there is not really a 'best' option; all choices would work more or less equally well.

A significant amount of digital design is reserved for verification and debugging. If for some reason you have problems with your circuit, try the following.

- Isolate the problem; you can just map the display decoder by connecting 4 inputs to the switches and the 7 outputs to the 7-segment display. In this way you can check if the display driver alone works correctly. If this is the case, the problem could be in the adder circuit, or in the interconnection between the two.
- If something is displayed on the 7-segment display, but it does not match the expectations, consider that all output LEDs are independent. Try to find out which LEDs function correctly (a, b, c etc.) and which ones do not. You should then find the place in the Verilog code that determines the output of the LEDs that are not working correctly.
- Check your bit ordering. The given constraints for port D are only valid if you followed the outlined ordering of bits (i.e. D(0) corresponds to LED A, D(1) to LED B, etc.)

Until now, we have designed combinatorial circuits. The outputs of these circuits were directly determined by their inputs. We use these circuits extensively, but they are not able to remember what has happened in the past: they have no memory, only the present time. Starting with the next exercise, we examine state holding circuits that can differentiate among different states and can move through different states depending on the input and the present state. These circuits are called finite state machines.

Digital Circuits Laboratory Report

Lab 3: Verilog for Combinatorial Circuits

Date		Grade
Group Number		
Names		Assistant

Part 1

Complete the truth table for the 4-bit to 7-segment decoder.

Display	S3	S2	S1	S0	A	B	C	D	E	F	G
0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	1							
2	0	0	1	0							
3	0	0	1	1							
4	0	1	0	0							
5	0	1	0	1							
6	0	1	1	0							
7	0	1	1	1							
8	1	0	0	0							
9	1	0	0	1							
A	1	0	1	0							
B	1	0	1	1							
C	1	1	0	0							
D	1	1	0	1							
E	1	1	1	0							
F	1	1	1	1							

Part 2

Attach the Verilog code of the decoder.

Part 3

(Confirmation required) Show the working circuit to an assistant.

Part 4

If you have any comments about the exercise please add them here: mistakes in the text, difficulty level of the exercise, anything that will help us improve it for the next time.