ELE 206/COS 306 Lab 3: Sequential Logic

Due Date: October 17th, 2018
Demonstration Required.

Tutorial Reading: Chapters 1, 2, 3, and 7
Tutorial Review Questions: 1.10, 1.12, 3.1, 3.2, 3.3, 3.4, 7.3

Introduction

In this lab, you will write and debug Verilog to simulate simple sequential circuits. This assignment will emphasize the different ways that a circuit could be written in Verilog. It is expected that you have completed the tutorial reading and tutorial review questions prior to beginning the lab. Your answers to the review questions should be a part of your write-up that is submitted at the end of the lab.

To begin, download and unzip *lab3.zip* from Blackboard, and follow the instructions in this document.

You may also wish to review the lecture slides or textbook sections covering single-bit memory storage blocks. This is covered in section 3.2 of the textbook.

Part 1: A simple latch

In lecture, various forms of latches and flip-flops were discussed. These are also discussed in textbook section 3.2. We will begin by implementing a latch in Verilog. The textbook and lecture both present a SR latch implemented using NOR gates. Here, we will implement a different kind of latch in structural Verilog using NAND gates as shown in Figure 3.1.

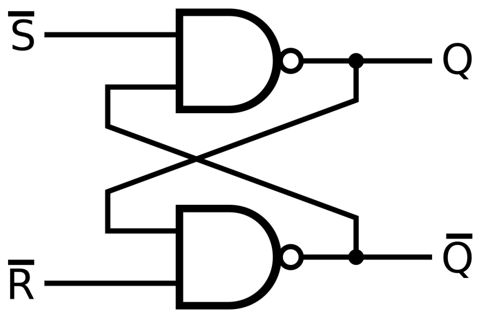

Figure 3.1: An $\bar{\text{S}}\bar{\text{R}}$ latch

You should be able to reason from the circuit schematic that when the $\bar{\text{S}}$ (not set) input of the latch is 0 and $\bar{\text{R}}$ is not, the output Q gets set to 1. When $\bar{\text{R}}$ (not reset) is 0 and S is not, Q will be reset to 0. We can see this is a sequential circuit and not merely a combinational one because there is some input value for which the output Q could be either 1 or 0, depending on previous inputs.

Write-up Question 1:
Show how the circuit is bistable when the input is $(\bar{\text{S}}, \bar{\text{R}}) = (1, 1)$, that is, the circuit could be in two different stable states with different output values. Describe how the latch could be operated to store one bit of information.

Write-up Question 2:
What happens when the input is $(\bar{\text{S}}, \bar{\text{R}}) = (0, 0)$? What would happen if the input instantaneously transitions from (0, 0) to the "store" input, (1, 1)? Would something be stored? What if the transition is not instantaneous, and one bit in the input transitions from 0 to 1 some time before the other?

Structural Verilog

Open the file *Latch_structural.v*. It has a module named `Latch` with two inputs `ns` (not set) and `nr` (not reset), as well as two outputs named `q` and `nq`. Recall that all of a module's ports are implicitly `wire`s, and can therefore be assigned to using continuous assignment, and can be used as values in expressions as well. Fill in the logic of the module to make it match what is shown in Figure 3.1. Use two `assign` statements, one representing each NAND gate.

Save the file *Latch_structural.v*. We have provided a simple testbench named *Latch.t.v* that simply provides a sequence of inputs to the `Latch`. Note how the testbench module instantiates the `Latch` and gives it a series of input values, using delays between switching input values. Save these two files in the same directory, and then compile the files together and view the simulation results.

```
iverilog -Wall -o Latch_test Latch_structural.v Latch.t.v
vvp Latch_test
gtkwave Latch_test.vcd
```

Look through the waveforms carefully to ensure that your latch is functioning correctly.

Write-up Question 3:
The simulation avoids the problematic transition from $(0,0)$ to $(1,1)$ discussed in the previous write-up question. The simulator is a single-threaded program running on your computer, meaning it cannot perform two computations in parallel. Instead, after each change in input value, it works its way through the circuit, evaluating changes in signal values one after another. Explain how different simulators might give different results when attempting to simulate the problematic transition in input values. You do not need to check your answer using simulation.

Behavioral Verilog

Open the file *Latch_behavioral.v*. This will be another implementation of a `Latch`, but now both outputs are declared to be `regs`. Fill in the `always` block with procedural assignment and any necessary code structures to make the module function as in the structural implementation: (`ns`, `nr`) = (1, 0), (`ns`, `nr`) = (0, 1), and (`ns`, `nr`) = (0, 0) are input values that can change the values of `q` and `nq`. As you found previously, the input transitioning from (0, 0) to directly to (1, 1) is problematic, so you can simply ignore this case, and assume that both inputs bits will not simultaneously change from 0 to 1. You can therefore also assume that whenever the input is (1, 1), the values of `q` and `nq` are stored and do not change.

Save the file *Latch_behavioral.v*. You can compile it with the same testbench as in the previous part and perform a simulation:

```
iverilog -Wall -o Latch_test Latch_behavioral.v Latch.t.v
vvp Latch_test
gtkwave Latch_test.vcd
```

The result of this simulation should be identical to the simulation of your structural Verilog implementation of a `Latch`.

Write-up Question 4:
In a large computer system, memory units must be occasionally replaced or upgraded, e.g. upgrading a hard drive in a data center. These memory units behave similarly, sometimes following an industry standard, but they may be implemented very differently at the gate-level. Similarly, in a larger circuit design, a designer may want to substitute one latch for another. This would need to be done without causing functional changes in the larger design. What conditions and/or restrictions on input behaviors should be maintained across the design in order for a designer to be able to successfully substitute an $\bar{\text{S}}\bar{\text{R}}$ latch for another?

Write-up Question 5:
What are the advantages and disadvantages of implementing this latch structurally and behaviorally? Which implementation do you think is best? Why?

Part 2: Larger Sequential Circuits

Figure 3.2 shows a larger sequential circuit. The output behavior of this circuit for different inputs is not obvious from the circuit schematic.

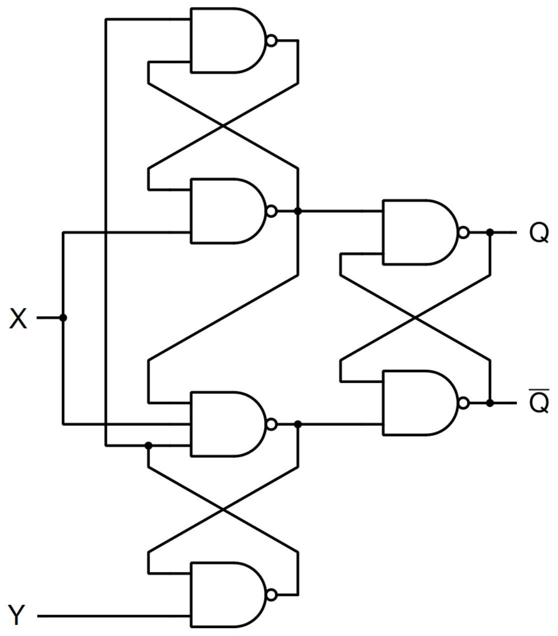

Figure 3.2: A mysterious sequential circuit schematic

Structural Verilog

Open the Verilog file named *MemBlock_structural.v* containing a module named `MemBlock` that will implement the circuit in Figure 3.2 using structural Verilog. It has two inputs named `x` and `y`, as well as two outputs named `q` and `nq`. You should be able to reduce the amount of logic you must write by using three instances of the `Latch` module from the previous part, and connect them with additional wires you may declare as necessary, as well as only a very small amount of additional logic. *Hint: the three-input NAND gate in Figure 3.2 is equivalent to two two-input gates connected properly.*

You may not declare any `reg` datatype in this file. You may declare more `wire`s and use continuous assignment or module instantiation to assign values to any `wire`s. You will receive partial credit on this section if you do not make effective use of three `Latch` instances in your circuit implementation.

To find out the behavior of this circuit, we can simply simulate it instead of trying to analyze it by hand. Create your own testbench for this circuit in a file named *MemBlock.t.v*. You may base your testbench on *Latch.t.v*, but you should add to the testbench in order to test more possible transitions between different input values, or to test different sequences of possible transitions, to better clarify the behavior of the circuit and answer the write-up questions. Use the `$dumpfile` preprocessor directive to save data to a file named *MemBlock_test.vcd*. After saving the files, compile your implementation of the circuit together with the testbench. If you used the `Latch` module in your *MemBlock_structural.v*, you should include the structural `Latch` implementation you wrote earlier in the compilation step. For example, you might use these commands:

```
iverilog -Wall -o MemBlock_test MemBlock.t.v MemBlock_structural.v
    Latch_structural.v
vvp MemBlock_test
gtkwave MemBlock_test.vcd
```

Write-up Question 6:
What 1-bit memory block that was discussed in lecture and in the textbook does this circuit behave the most like? Identify one of the inputs of this circuit as either an enable or clock signal. To support your answer, include one or more screenshots of waveforms in GTKWave recorded from the simulation of the circuit with your testbench.

Write-up Question 7:
In terms of the number of two-input gates that make up the circuit, compare this circuit with the memory block that it behaves similarly to that was discussed in lecture and in the textbook. Which likely uses fewer transistors?

Behavioral Verilog

Finally, open the file *MemBlock_behavioral.v*. Fill in the code to make this behave identically to the other `MemBlock` module, as far as its ability to load and store values based on the inputs. You should find a very simple solution from carefully reading Chapter 3 of the Verilog tutorial. In particular, carefully consider what to put in the sensitivity list to make the `always` procedure run at the correct times, based on your answer to write-up question 6. This module should not contain any continuous assignment or instantiate any other modules.

You should test your module with your testbench and verify that the simulated behavior is the same as with your other `MemBlock` implementation, with the possible exception of the beginning of a simulation, before the `MemBlock` begins storing a value determined by the inputs supplied by the testbench. You should be able to do this using the following commands:

```
iverilog -Wall -o MemBlock_test MemBlock.t.v MemBlock_behavioral.v
vvp MemBlock_test
gtkwave MemBlock_test.vcd
```

Write-up Question 8:
Please leave a bit of feedback regarding this lab. How much time did it take, how difficult was it, did you get stuck anywhere? There are no wrong answers here – we'll be using the feedback to adjust this lab for the future.

Demonstration, Write-Up, and Submission

Demonstration

For your demonstration this week, visit the F114 lab during a lab TA session to discuss your code for both *MemBlock_structural.v* and *MemBlock_behavioral.v* with a TA and show the behavior of either one of them in GTKWave. You will need to explain to the TA what determines when a `MemBlock` is storing a value and when it is loading a new value.

Write-Up

For your write-up, complete the tutorial review questions assigned at the beginning of the lab and answer the 8 numbered questions from the text briefly but completely. Feel free to create the write-up in whatever program you want to use, but save and submit it as a PDF. Name your PDF *netid_lab3_writeup.pdf*, with *netid* replaced with your Princeton NetID.

Submission

Delete *Latch_test*, *MemBlock_test*, and any *.vcd* waveform files from the */lab3* directory. You can do this via your file browser or with one of the following commands.

On OS X and Linux:

```
rm -f *.vcd Latch_test MemBlock_test
```

On Windows:

```
del *.vcd Latch_test MemBlock_test
```

Then create a *.zip* archive of that directory. Make sure to name it as *netid_lab3.zip*, with *netid* replaced with your Princeton NetID.

On OS X and Linux, simply change directories in your terminal to the the directory that contains the */lab3* directory and then execute this command:

```
zip -r netid_lab3.zip ./lab3
```

On Windows, simply open up File Explorer and locate the */lab3* folder. From Command Prompt, you can open your current directory's parent in File Explorer by issuing this command:

```
explorer.exe ..
```

Right-click on the */lab3* folder and select "Send to", then "Compressed (zipped) folder". This will create a *.zip* file.

For reference, here is a checklist of files that must be in your *.zip*:

- *Latch_structural.v*
- *Latch_behavioral.v*
- *MemBlock_structural.v*
- *MemBlock_behavioral.v*
- *MemBlock.t.v*

Finally, submit the following to Blackboard:

- *netid_lab3.zip*
- *netid_lab3_writeup.pdf*