ELE 206 Lab 1 - Oliver Schwartz

Tutorial Review Questions

1.1 Describe the fundamental differences between a program written in a high-level language e.g. Java, and a program written in Verilog. How does your approach to program design change when programming in Verilog?

Verilog is a HDL (hardware description language) which is used to describe a digital circuit. Some of the fundamental differences include: concurrency - Verilog allows for concurrent computation, where statements and assignments are executed simultaneously (in parallel). Languages like Java work differently, executing statements in sequential order rather than in parallel. The level of complexity between Verilog and Java is vastly different: Verilog allows for bit-level control of hardware, mapping on/off values to individual sets of wires and registers. Java allows for a higher level of abstraction, and a Java programmer does not have to deal with hardware assignment.

When programming in Verilog, program design changes because you are not dealing with abstract programming constructs like classes and objects. Instead, you are implementing a physical circuit with a programming language. However, things like logical constructs are still present in both languages (if, if-else, else, switch statements).

1.2 What does it mean for a wire to be* driven*? What value does a wire take when it is not being driven? What about when it is being driven by more than one driver?

For a wire to be 'driven' means for a wire to have a digital value (logical 0 or 1) - in physical terms this corresponds to a voltage. If a wire is left to float (not driven) it can take the value of either Z (high impedance) or X (indeterminate). If a wire is being driven by more than one driver, than the wire will take the value 1 if any of the drivers are 1, and 0 otherwise. If a wire is connected to a tri-state buffer which is enabled, it takes the value of the input. If the buffer is disabled, then the wire takes on a high-impedance value.

1.8 What is the difference between continuous and procedural assignment in Verilog? Which one describes hardware more naturally? Why?

Continuous assignment ensures that if a wire's value is derived from the output of some other grouping of wires (say B and C), whenever B or C change, A will change instantaneously. An example of continuous assignment is: **wire A = B && C;**

Procedural assignment is also used to create combinational logic assignments (however, the target for assignment is a register datatype, but in a procedural block this is equivalent to a wire). Procedural assignment takes place within an **always** block. Procedural assignment describes hardware more naturally, as it provides a well organized set of conditionals in case a change in the system takes place. Procedural assignment also allows for the use of if/elseif/else/switch conditional constructs which is more easily translated into understanding the physical circuit design being represented.

1.11

a) Classify each module as either a structural or behavioral model.

Module A uses continuous assignment and wire data types, and is therefore a structural model. Module B uses procedural assignment and reg data types, and is therefore a behavioral model.

b) How does the execution of code differ between these two implemenations?

Implementation A uses continuous assignment, whereas B uses procedural assignment (contained within the for loop). Because of this, d is a wire in A whereas in B it is a register (although for all intents and purposes the register in B can be thought of as a wire).

c) The sensitivity list for the* always *procedure in* SimpleCircuitB *contains four signals. Let's say the value of the signal* a *changes and the procedure is triggered. Consequently, the value of* d *changes, but is the procedure triggered again? Does the signal* d *need to be in the sensitivity list?

If a changes, the procedure is triggered. This changes the value of d. However, although d is in the sensitivity list, this change does not trigger the procedure again because a change within an **always** block cannot trigger the same block. **D** does not need to be in the sensitivity list, as d's value is a function of **a** and **b,** so only a, b, and c need to be in the sensitivity list.

d) How might you rewrite the sensitivity list to be better?

As above: **D** does not need to be in the sensitivity list, as d's value is a function of **a** and **b,** so only a, b, and c need to be in the sensitivity list.

e) For this circuit, which style of writing seems more intuitive? Why?

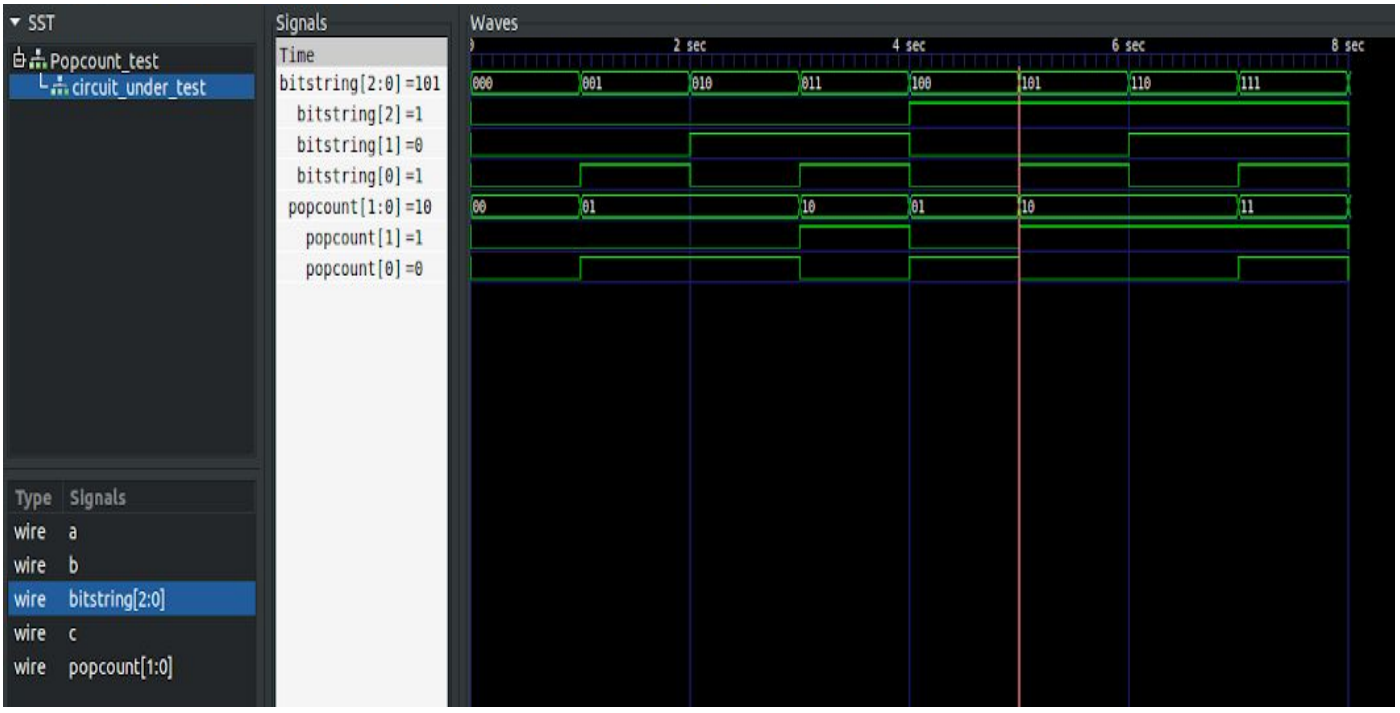
Procedural assignment (b) seems more intuitive, as the **always** term implies that this section of code is triggered whenever the inputs change - whereas the continuous assignment does not make this obvious (although functionally they are identical).

2.1 Explain why breaking down complex digital circuit designs into* modules *can be useful for hardware designers.

Hardware designers are often working on large, complex digital circuits, which are massive and have many internal components. For example, all the wires and registers for an entire CPU in a single file location would lead to duplication (as there are many adders/AND/OR/NOT gates in a CPU). This would create very long and unreadable code. Therefore, modules are useful by preventing code duplication. Modules also allow hardware designers to break up a complex system into small parts which are easier to test and debug.

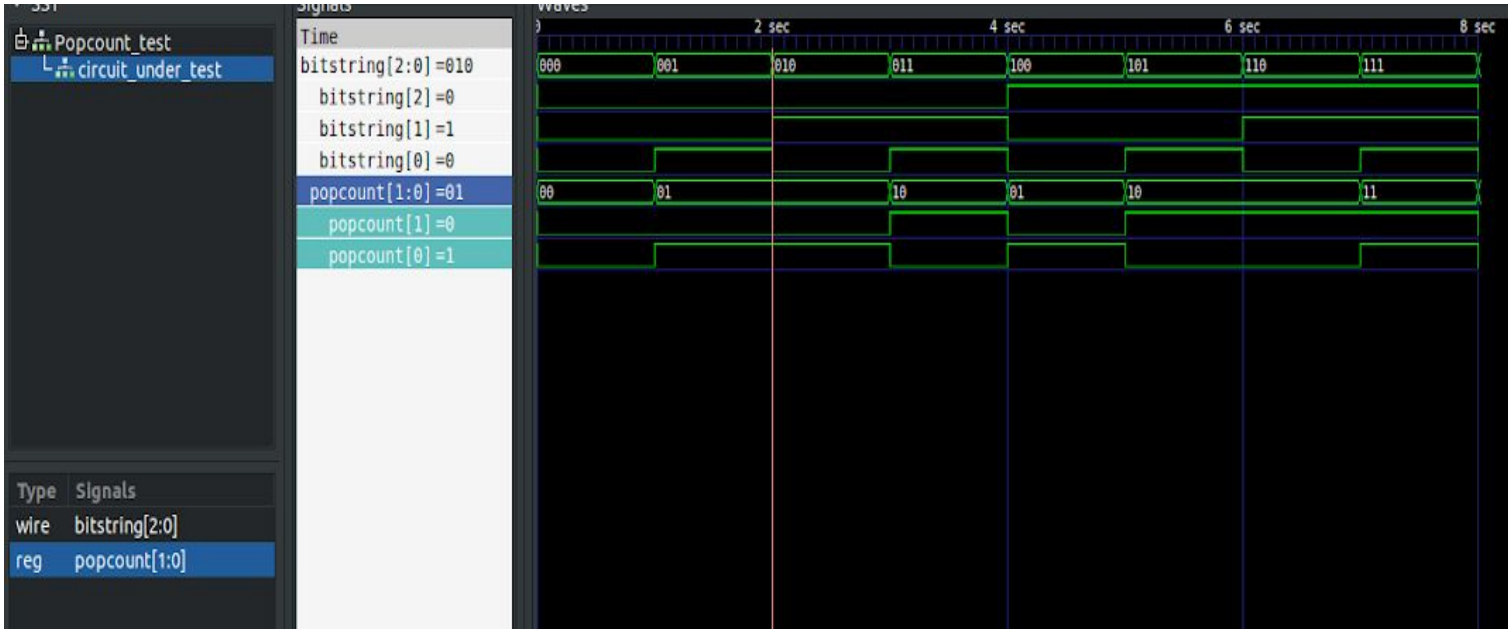
Lab Write-up

Question 1



At the point shown, the bitstring represents the decimal value 5 in binary (101) and popcount outputs 10 (2 in decimal) as there are 2 1's in 101. The waveforms for the individual bits do not convey any additional information than the values shown for the multi-bit signal if it were not expanded (the condensed binary form is all the information that is necessary; the individual waveforms for the 3 wire inputs can simply be inferred as they are digital values).

Question 2



The values in the “waves” pane shown for the multi-bit vectors indicate the circuit is computing the correct value at all points in time because the value of both registers in the **popcount** vector are correct at all points in time. In other words, when **bitstring** has 3 1’s, **popcount’s** registers take the value 11 (or 3 in decimal). When **bitstring** has 2 1’s, **popcount** takes on the value 10 (or 2 in decimal). Likewise, for one 1 and zero 1’s, **popcount** takes on the values 01 and 00 respectively (1 and 0 in decimal).

Question 3

For this particular module, you might prefer to write it in structural Verilog as this is fewer lines of code. Considering the module is so simple, it can be easily understood in structural Verilog. Behavioral Verilog would be more useful for a larger and more complex circuit.

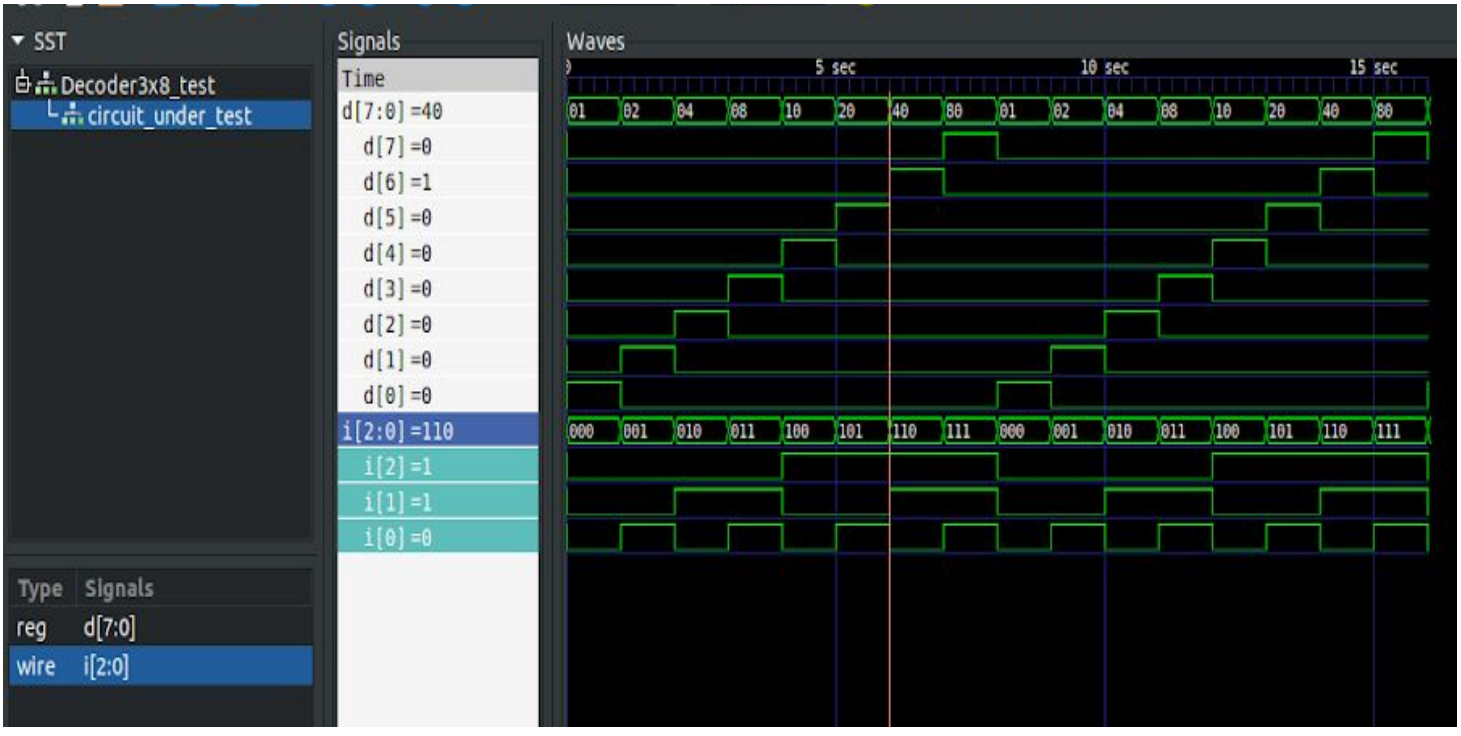
Question 4

- Error Line 3: Syntax error. There was a missing square bracket, **output 7:0]** should have been **output [7:0]**
- Error Line 6: missing semi-colon at end of line.
- Error line 8: missing sensitivity list. **always begin** should have been **always @(*) begin** (or some variation with variables in the parentheses).
- Error line 34: **end** should have been **endcase** as it corresponds with the **case** logical construct on line 9.

Question 5

A 3 to 8 decoder converts a 3 bit binary input into a 1 bit (decimal) output. This code does not successfully implement the decoder combinational circuit. This can be observed in GTKWave, specifically when the input changes from 110 to 111. When this change occurs, there is no change in the output (and therefore the decoder cannot decode all possible values from 000 to 111 because it does not have unique output for each input - i.e. 110 cannot be distinguished from 111)

Question 6



Question 7

There are benefits to both structural and behavioral Verilog in this instance. For structural Verilog, there are fewer lines of code. However, this comes at a price: it is less obvious how the decoder is working in structural Verilog. On the other hand, behavioral Verilog requires more lines of code. However, behavioral Verilog makes it more obvious what is happening: with the **case** logical construct, it is clear that each input is parsed as a decimal number, and the matching output wire is then given the binary value of 1. This is easier to understand than the series of midterms in the continuous structural Verilog assignments.

Question 8

Feedback: this lab took ~3 hours. The majority of the time was spent reading the introduction to Verilog document. It was not too difficult, and I did not get stuck.