Due date: Thursday February 15 11:59:59pm EST.

Turning in the lab: This lab has written and electronically-submitted components. To turn in the written portion of this lab (Exercises 1–7 and Discussion Question 1) you can either hand in a paper copy of your answers in lecture on the due date, or you can e-mail a scanned copy of your answers to 6s084-staff@csail.mit.edu. To turn in the Bluespec portion of this lab, commit and push your changes you made to your git repository.

Check-off meeting: After turning in this lab, you are required to go to the lab for a check-off meeting within a week of the lab’s due date. See the course website for lab hours.

1 Boolean Algebra

Exercise 1 (10 points): Simplify the following Boolean expressions by finding a minimal sum-of-products expression for each one:

- \(ac + b + c\)
- \((a + b)c + \overline{c}a + b(\overline{a} + c)\)
- \(a(b + c)(b + a(c + b))\)
- \(a(b + c(d + ef))\)

Exercise 2 (10 points): There are some Boolean expressions for which no assignment of values to variables can produce True \((e.g., a\overline{a})\). Those Boolean expressions are said to be non-satisfiable. Are the following Boolean expressions satisfiable? If the expression is satisfiable, give an assignment to variables that makes the expression evaluate to True. If the expression is non-satisfiable, prove it.

- \((a + b)c + \overline{c}a + b(\overline{a} + c)\)
- \((x + y)(x + \overline{y})(z + \overline{y})(y + \overline{z})\)
- \((x + y + z)(x + y + \overline{z})(x + \overline{y} + \overline{z})(x + y + \overline{z})(x + \overline{y} + z)(x + \overline{y} + \overline{z})(x + \overline{y} + \overline{z})(x + y + z)\)
- \(xyz + xy\overline{z} + x\overline{y}z + xy\overline{z} + x\overline{y}z + xy\overline{z} + x\overline{y}z + x\overline{y}z\)

Exercise 3 (10 points):
1. Write \(-13\) and \(16\) in 8-bit two’s complement representation.
2. Show the steps of the binary addition of those numbers.

Exercise 4 (5 points): We proved during recitation that \(x = b_m\ldots b_0\) is a multiple of 4 if and only if \(b_0 = 0\) and \(b_1 = 0\).

In the same spirit, let \(x = h_n\ldots h_0\) be a number written in hex (base 16). Find a simple condition on \(h_0\) that characterizes when \(x\) is a multiple of 4.

Exercise 5 (10 points): As you have seen, \(\{\text{AND, OR, NOT}\}\) are functionally complete, \(i.e., these Boolean functions suffice to build any other Boolean function by composition.\) In this exercise you will show that the NOR function, \(\text{NOR}(x, y) = \overline{x + y}\), is also functionally complete by expressing AND,OR, and NOT using NOR.

- Build a NOT function with NOR functions.
- Build a AND function with NOR functions.
- Build a OR function with NOR functions.
- As an example, find a Boolean expression equivalent to \(c((a + b)c + \overline{c}a + b(\overline{a} + c))\) that uses only NOR functions.
Exercise 6 (5 points): Write a Boolean expression \( vote(v_1, v_2, v_3) \) that returns 0 if a majority (i.e., more than half) of the inputs are 0, and returns 1 otherwise.

Exercise 7 (10 points): Let \( f \) and \( g \) be two Boolean functions with four inputs: \( f(a, b, c, d) \) and \( g(a, b, c, d) \). Let’s define \( h(a, b, c, d) \) to be \( f \) if there is an even number of 1s in the inputs, and \( g \) otherwise. Write an expression for \( h(a, b, c, d) \) depending on \( g(a, b, c, d) \) and \( f(a, b, c, d) \).

2 Functions in BSV

Let’s first examine Multiplexer.bsv.

```haskell
function Bit#(1) multiplexer1(Bit#(1) sel, Bit#(1) a, Bit#(1) b);
    return (sel == 0)? a : b;
endfunction
```

The first line begins a definition of a new function called \texttt{multiplexer1}. This multiplexer function takes several arguments that will be used in defining the behavior of the multiplexer. This multiplexer operates on single bit values, the concrete type \texttt{Bit#(1)}. Later in the course we will learn how to implement functions that can handle arguments of any width.

This function uses the \textit{ternary operator} construct in its definition, which takes two inputs and selects between them using a conditional expression. This syntax, \texttt{cond ? a : b}, is common in many languages like C. Its equivalent in Python is \texttt{a if cond else b}. The single return statement constitutes the entire function. The \texttt{endfunction} keyword completes the definition of our multiplexer function.

Simple code such as the multiplexer’s can be defined at a high level, relying on tools to produce an efficient circuit implementation. However, because hardware compilation is a difficult, multi-dimensional problem, tools are limited in the kinds of optimizations they can do.

You should be able to compile the module by typing: \texttt{make mux1; ./simMux1}

Exercise 8 (5 points): Using the \texttt{and1}, \texttt{or1}, and \texttt{not1} gates, implement the function \texttt{multiplexer2} in Multiplexer.bsv. This function should use a 1-bit select signal to select between two 2-bit inputs. Test your code by typing \texttt{make mux2; ./simMux2}.

Discussion Question 1 (5 points): How many gates are needed? Please include your solution in the written portion of the lab.