cs281: Computer Organization

Lab2 Prelab

The purpose of this prelab is to introduce some of the fundamentals of *Combinational Logic Design*, preparing us for using the breadboards to build circuits designed in this Prelab. The prelab will take us through a number of examples of the design process and the relationship between digital logic and Boolean expressions and then the lab itself will focus on implementing and verifying a digital logic circuit.

1 Combinational Logic Design

Combinational Logic Design refers to the use of logic gates that, when combined, perform some boolean function. The boolean function results in output(s) that are dependent only on the input values and are independent of previous input values or states. In other words, the current inputs completely determine the output(s).

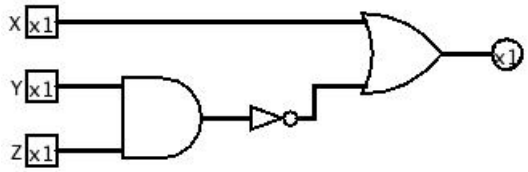
1.1 Fundamentals

Expressions in boolean algebra consist of constants, boolean variables, and operators, and additional precedence/order of operation can be enforced by using parenthesis. For the purposes of digital logic, we use 0 and 1 to represent boolean *false* and *true*, respectively. We primarily use single letter variables, sometimes subscripted when we need a set of variables, and each variable represents a value of either 0 or 1. For the boolean logical operators of *and* and *or*, we borrow the arithmetic notions of product and addition, and use symbols of $\cdot$ and $+$, respectively. We also borrow from arithmetic algebra and say that two variables placed next to each other implicitly have an *and* operation between them. For logical negation, we use a line over the boolean expression to be negated. So the expression:

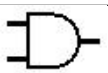
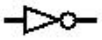
$$X + \overline{YZ}$$

represents X *or*'d with the negation of the composite expression (Y *and* Z). In boolean algebra, negation has precedence over *and*, which has precedence over *or*, so in $X\overline{Y}+Z$, the $\overline{Y}$ is evaluated first, followed by the *and* with X, followed by the *or* with Z.

In both the expressions of boolean algebra and in their realization in digital logic, the *variables* represent the *input* to the expression or the circuit. A digital logic realization of a boolean algebra expression is then remarkably straightforward. Given digital logic elements (called gates) that implement *and*, *or* and *not* with voltages representing logical 0 and 1 (typically Ground and +5 volts, respectively), then there is a direct mapping of the boolean algebra to the circuit, using the precedence of the boolean algebra to feed the order of the gates in the circuit. We often depict a circuit with the inputs on the left or the top, and work to the right and/or down toward the result of the expression (the boolean *output*). So the expression above, represented as a circuit is depicted below.



where the lines represent wires and the symbols are as follows:

	Represents an input pin, where the notation inside the box gives the number of input bits (1 bit input in this example)
	Represents an output pin, where the notation inside the circle gives the number of bits (1 bit of output in this example)
	Represents an *and* gate, with the two inputs on the left and the output on the right
	Represents an *or* gate, with the two inputs on the left and the output on the right
	Represents a *not* gate, with the input on the left and the output on the right

At an abstract level, we can describe the combinational digital logic design/realization process as composed of two steps:

1. Derive a set of boolean expression whose variables are the inputs to the circuit and each expression defines a single boolean output. This set of boolean expressions express the desired functionality of the combinational circuit. Many different boolean expressions can represent equivalent functionality, and we typically want as simple a set of boolean expressions as possible.

2. Given the set of boolean expressions, realize the circuit in gates and wires, allowing inputs and outputs to be manipulated and checked.

This prelab leads you through the learning of some initial tools and mechanisms to achieve Step 1, while the Lab will allow you to practice with Step 2. It is through the design of combinational circuits that, given a binary representation of data items in a computer and a program running on a computer, we can achieve *computation*.

1.2 Truth Table

A *truth table* is a mechanism for complete specification of one or more boolean outputs for a set of boolean inputs. Along with the outputs, we may sometimes include columns for subexpressions, and these have no effect on the number of rows in the truth table. The number

of rows in a truth table is dependent only on the number of boolean input variables. If n is the number of boolean input variables, the truth table should have 2^n rows. You should have already seen truth tables for specifying the values of boolean expressions given a set of boolean input variables in your introductory computer science class, so this should be review. Consider the following truth table with two boolean input variables A and B, with output Y:

row	A	B	Y
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

Note that we will generally not annotate with row numbers, included here for ease of reference. Also note that when we consider the order of the rows and the corresponding possible values of the input variables A and B, the order follows a binary counting order when the values of A and B are interpreted as a binary value. This row order is a *standard order*, and you should follow the same.

Q1 : Write down an English sentence characterizing the output, Y, as a boolean logical combination of the inputs.

Q2 : Is Y equivalent to any of the basic boolean operators that you know?

Q3 : Fill in a boolean algebra expression defining Y:

$Y =$

Q4 : Draw a gate realization for Y based on your answer to the previous question.

Now answer Q1, Q3, and Q4 (and label as Q5, Q6, Q7) for the following truth table:

row	A	B	Y
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	0

Q8 : How many *different* truth tables are possible for the case with exactly two input variables? Justify your answer.

1.3 English to Truth Table

Sometimes a designer is given a natural language description of some desired combinational output given a set of boolean inputs. One could attempt to translate directly to boolean

expressions, but this can be error prone, and so a complete specification of the desired logic can be attained by "translating" from the natural language to a truth table. We will practice that here, and will use problems involving three input variables.

Q9 : Suppose we have three boolean inputs, denoted X_2, X_1, X_0. Fill in the following truth table where the single output Y is determined as follows: if X_0 has value 0, then Y takes on the value of X_2, but if X_0 has value 1, then Y takes on the value of X_1.

row	X_2	X_1	X_0	Y
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

Q10 : Suppose we have three boolean inputs, denoted X_2, X_1, X_0. Fill in the following truth table where the two outputs Y_1 and Y_0 are determined as follows: if we interpret Y_1Y_0 as a two bit sequence, we want its value to be determined as the *arithmetic sum* of the three single bit inputs $(X_2 + X_1 + X_0)$.

row	X_2	X_1	X_0	Y_1	Y_0
0	0	0	0		
1	0	0	1		
2	0	1	0		
3	0	1	1		
4	1	0	0		
5	1	0	1		
6	1	1	0		
7	1	1	1		

1.4 Truth Table to Boolean Expression

The missing step, at this point, is how, in general, to go from a truth table to a boolean expression. Consider your answers to Q3 and Q6 from earlier. If we focus on a single row of the truth table in which the boolean output variable is 1, we should have been able to

arrive at a boolean expression (a *term*) involving an *and* of all the input variables or their individual negation. Such an expression evaluates to 1 (true) exactly when the values of the inputs correspond to their specified value in the row.

An example might help clarify. Consider row 3 from Q9. Since X_0 is 1, Y should have output 1, since that is the value of X_1. We desire a boolean term that is true (or 1) when X_2, X_1, X_0 have the specified values, so when X_2 is 0 *and* X_1 is 1 *and* X_0 is 1. The desired term is $\overline{X_2}X_1X_0$. Procedurally, we construct an *and* and use the negation of an input variable where the value in the row is 0, and the positive sense of the input variable where the value in the row is 1. When this boolean expression is true, the inputs correspond to that particular row of the truth table. We call a term corresponding to a true/1 row of the truth table a *minterm*.

Q11 Write down all the terms corresponding to the rows where Y is 1 in your truth table for Q9.

Q12 Write down all the terms corresponding to the rows where Y_1 is 1 in your truth table for Q10.

Q13 Write down all the terms corresponding to the rows where Y_0 is 1 in your truth table for Q10.

Consider again your answers to Q1, Q3, Q5, and Q6. When we had just a single row with output 1, the English and the boolean algebra expression consisted of just the term corresponding to the "true" row (i.e. the minterm). When more than one row had output 1, the English and the boolean expression had to capture that the output was true when the inputs corresponded to either one row *or* another row. This same reasoning generalizes when we have more input variables and more rows in the truth table and more rows with an output of 1. The output is true when any of the cases of the combination of the input variables correspond to that true row. So, in general, an expression for an output can be constructed as the logical *or* of all the minterms. Since a minterm is an *and* and we use product for *and* and we use addition of *or*, this is a standard form known as *Sum of Products* (SOP).

Q14 Fill in the SOP boolean expression defining Y from your truth table for Q9.

$Y =$

Q15 Fill in the SOP boolean expression defining Y_1 from your truth table for Q10.

$Y_1 =$

Q16 Fill in the SOP boolean expression defining Y_0 from your truth table for Q10.

$Y_0 =$

1.5 Simplification

Consider the following definition of boolean output Y:

$$Y = X_2\overline{X_0} + X_1X_0$$

Q17 From the given definition, fill in the truth table corresponding to the definition of Y. I have added columns for the values of the terms in the expression to help the process.

row	X_2	X_1	X_0	$X_2\overline{X_0}$	X_1X_0	Y
0	0	0	0			
1	0	0	1			
2	0	1	0			
3	0	1	1			
4	1	0	0			
5	1	0	1			
6	1	1	0			
7	1	1	1			

Q18 Now fill in the SOP boolean expression defining Y from your truth table.

$Y =$

Q19 Look carefully at the given definition of Y and your SOP-derived definition of Y. Do you believe them to be equivalent? If so, try and reason and articulate how the simpler form might be logically derived from the more complicated form. If not, explain why you believe they are not equivalent.