cs281: Introduction to Computer Systems

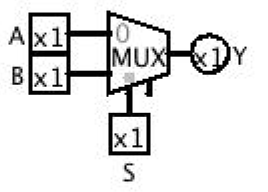
Lab02 – Basic Combinational Circuits: The Mux and the Adder

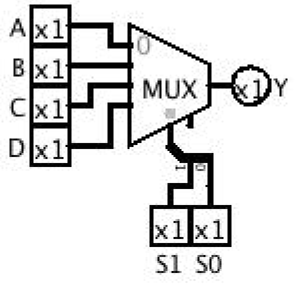
Overview

The objective of this lab is to understand two basic combinational circuits — the multiplexor and the adder. It also completes the second step of the design process, wherein you go from boolean algebra expressions that solve a particular "computation" defining outputs for each expression, realize the boolean expression with gates and wires, and also use the Arduino to assist in testing and validating your circuits. This Lab assumes you have already completed the associated Prelab. You should answer any questions asked below and turn in your lab response at the end of the lab. For each completed circuit, you will also demonstrate its correct operation for your instructor or TA.

Multiplexor

A *multiplexer* (aka *mux*) is a combinational circuit that selects one of its n input lines and provides it on the output. A 1 bit 2-1 mux has an n of 2, has 1 output, and the *width* of the input lines and output line are a single bit wide. Every mux requires additional input known as the *select* lines, which control which of the n inputs is selected for the output. The figure below shows, on the left, a 1-bit 2-1 mux, with its 2 selectable input lines labeled A and B, its select line labeled S, and its output labeled Y, On the right is a 1-bit 4-1 mux with its 4 selectable input lines labeled A, B, C, D, its select lines labeled S_1, S_0, and its output labeled Y.





Q1 Consider the truth table from Q9 of the prelab. Is there a relationship between the boolean function described in Q9 and the boolean function of a mux as described above? If so, precisely describe the mapping between the inputs of one and the inputs of the other.

Q2 Given the description and picture of a 1 bit, 4-1 mux, how many rows would be present in a truth table for this function?

Q3 A partial truth table is given below. For the given rows/inputs, fill in the corresponding values of Y.

row	A	B	C	D	S_1	S_0	Y
0	0	0	0	0	0	0	
1	0	0	0	0	0	1	
2	0	0	0	0	1	0	
3	0	0	0	0	1	1	
4	0	0	0	1	0	0	
5	0	0	0	1	0	1	
6	0	0	0	1	1	0	
7	0	0	0	1	1	1	
8	0	0	1	0	0	0	
9	0	0	1	0	0	1	
10	0	0	1	0	1	0	
11	0	0	1	0	1	1	
12	0	0	1	1	0	0	
13	0	0	1	1	0	1	
14	0	0	1	1	1	0	
15	0	0	1	1	1	1	

The adept reader should realize that the truth tables from Q9 of the prelab, Q17 of the prelab, and the 1 bit 2-1 mux described above are all describing the same boolean function. When we realize a function as a digital logic circuit, we want to use as few gates as possible, and this translates to the simplest boolean expression equivalent to the desired function. For the mux, the equation given in section 1.5 of the prelab gives us this simplest form, and so the next section of this lab will build and test this 2-1 mux circuit.

$$Y = A\overline{S} + BS$$

Breadboard Initial Setup

For the purposes of the remainder of the mux portion of this lab, we will use A, B, and S to refer to the 3 inputs to the circuit, and Y to refer to the output. We also need to refer to elements of the breadboard, so S_1 through S_8 will refer to the breadboard's *Logic Switches* at the lower left side of the board.

1. Pick three of the 5 pin "rows" on the breadboard to be designated for the three inputs A, B, and S. Wire Logic Switch S_1 to A, S_2 to B, and S_3 to S.
2. Wire A to Logic Monitor/Logic Indicator 1, B to Logic Indicator 2, and S to Logic Indicator 3. This will give us the ability to visualize the inputs to the mux circuit.

3. Designate another row on the breadboard to be Y, and wire Y to Logic Indicator 8 (to give separation from the 3 inputs).

4. Obtain from your instructor or TA three logic chips, a 7404, 7408, and 7432 and insert all three on the breadboard such that the input pins are all independent of each other and of your designated rows for A, B, S, and Y. Make sure when you place the chips that the physical "notch" on the chip is oriented to the North/top of the breadboard.

5. Using the IC schematic for each of the three chips, find the pins that feed GND and Vcc (+5 V) to the chip and wire these to GND and +5V respectively.

The 2-1 Mux

On the prepared breadboard, wire the following boolean algebra expression for Y:

$$Y = A\overline{S} + BS$$

From the IC schematics, we can see that the 7404 chip provides six independent inverters (NOT gates), the 7408 provides four AND gates, and the 7432 provides four OR gates, so you will not need any more than the three provided chips. You will have to figure out the correct pins on the chips to use for inputs and outputs to realize the circuit.

Once you have completed the circuit, use the manual switches to check the output given different input combinations and convince yourself of its correct operation. If it does not function properly, work your way from the original inputs through intermediate wires (by connecting to another Logic Indicator) to track down the problem.

Arduino-based Testing/Validation

Using manual switches to drive the inputs of circuits and visually checking the outputs and comparing to expected values in a truth table is a cumbersome and error prone approach to the testing and validation of a circuit. So we will make use of the programmability of the Arduino and use it to drive the circuit through the input combinations and to check and validate the output of the circuit.

If you have not already done so, get an Arduino controller, and hook up your laptop, running the Arduino Sketch IDE, to the Arduino via the USB cable.

1. Wire GND and +5V on the breadboard to the Arduino.

2. Remove the wires from the Logic Switches S_1, S_2, S_3 to A, B, S and instead wire Arduino pins 11 to A, 12 to B, and 13 to S on the breadboard.

3. Wire Y on the breadboard to Arduino pin 10.

4. From the course web page or this PDF, copy the MuxValidate sketch below into the Arduino IDE as a new sketch, and *Verify* (i.e. compile) the sketch.

```
const int A[] = {0, 0, 0, 0, 1, 1, 1, 1};
const int B[] = {0, 0, 1, 1, 0, 0, 1, 1};
const int S[] = {0, 1, 0, 1, 0, 1, 0, 1};

const int Y[] = {0, 0, 0, 1, 1, 0, 1, 1};

const int WAIT0 = 300;
const int WAIT1 = 2000;

int index = 0;
int x;

void setup() {
  Serial.begin(9600);
  pinMode(11, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);

  pinMode(10, INPUT);
}

void loop() {
  digitalWrite(11, A[index]);
  digitalWrite(12, B[index]);
  digitalWrite(13, S[index]);

  delay(WAIT0);
  x = digitalRead(10);

  Serial.print(index);
  if (x == Y[index]) {
    Serial.println(": OK");
  } else {
    Serial.println(": BAD");
  }

  delay(WAIT1);
  index = (index + 1) % 8;
}
```

5. Perform a code walk-through and explain to your lab partner the logic of the sketch.

6. Upload the sketch and validate your circuit. You can see the validation output from the `Serial.println()` invocations by clicking the document-looking icon in the top right of the Arduiono sketch window. This opens a window that displays any `Serial.println()` output generated onboard the Arduino and sent (through the USB interface) back to the laptop. If the circuit does not work properly for all input combinations, debug your circuit. Because the Arduino is going to continue changing the inputs, you may need to go back to the switch-based inputs to give yourself the control you need to debug.

   Q4 Demonstrate the circuit for the instructor or assistant.

Full Adder Sum Output

A *full adder* is a circuit with three single bit inputs, often labeled A, B, and C_{in} that performs the computation for one "column" of the binary addition process – add the two 1-bit inputs A and B, along with a possible *carry in*, C_{in}, resulting in two output bits, the *sum* bit, labeled S, and the *carry out* bit, labeled C_{out}. Described slightly differently, the prelab definition of the truth table in Q10 is the same *full adder* definition, where X_2 and X_1 are A and B, X_0 is C_{in}, Y_1 is C_{out}, and Y_0 is S.

Circuit Design

Q5 Rewrite the truth table with inputs and outputs as described above:

row	C_{in}	A	B	C_{out}	S
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		

Q6 Fill in the boolean expression defining the sum bit, S, in terms of A, B, and C_{in} in SOP form:

$S =$

Q7 : If we assume 2-input AND and OR gates, and single bit NOT gates, how many gates would you use to create an implementation of S?

Q8 : Does your answer change if you look for common sub terms (such as the negation of an input used in multiple places in Y)?

Q9 : In the above, we need to AND together more than 2 input variables (or their negation), and then we need to OR together more than 2 of these AND-terms. Draw a picture of a digital logic circuit that accomplishes this only using AND and OR gates with 2 inputs each.

The focus of the next lab, lab3, is one of simplification, so that we can reduce the number of gates required for any given combinational circuit. In the remainder of this section, we will lead you through a different means of simplification and focus our energies on the sum (S) bit. This will expose us to an additional boolean logic gate and will also make an implementation of the S bit much more manageable for the purposes of the current lab.

Q10 : We can draw a partial truth table that includes just A and B as inputs and is the truth table for the case where C_{in} is 0. This partial truth table corresponds to the first four rows of the full truth table above. Fill in this truth table:

row	A	B	S for $C_{in} = 0$ (denote by X)
0	0	0	
1	0	1	
2	1	0	
3	1	1	

Q11 : Look carefully at this truth table. What boolean function has this characteristic that it is 1 if one or the other of the inputs is 1, but is 0 if both are 0 or if both are 1?

Q12 : Look at the values of S in the original truth table for the last four rows (when C_{in} is 1). Given your characterization from the last question, describe in English what is happening when C_{in} is 1.

Q13 : If we denote by the variable X the result of this two-input truth table, we can imagine a sub circuit that realizes X, dependent only on A and B, and can then use X in an aggregate circuit to solve a "bigger" problem. Say that in this new truth table we have input C_{in} and X, and we are still designing a circuit to compute S. Using your original truth table and using the derived X instead of the individual inputs A and B, we define our aggregate circuit:

row	C_{in}	X	S
0	0	0	
1	0	1	
2	1	0	
3	1	1	

So row 0 in this truth table corresponds to both rows 0 and 3 in the original truth table, since we have the same resultant value based on C_{in} and X. Row 1 corresponds to both rows 1 and 2, and so forth. If the multiple rows did not yield the same value for each of these cases, we might not be able to replace two input variables (A and B) with our computed value of X.

Q10 : Look carefully at this truth table. What boolean function captures S in terms of C_{in} and X?

Q11 : Based on the two truth tables, we should now be able to define S in terms of C_{in} and A and B:

$S =$

Q12 : If we assume 2-input gates, how many gates would you use to create an implementation of S?

Q13 : We can double check our derivation above by using a truth table to prove the equivalence on your definition of Y. Fill in the columns of the following truth table:

row	C_{in}	A	B	S	$A \oplus B$	$C_{in} \oplus (A \oplus B)$
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			

Q14 :Argue the equivalence between the original S (which could be realized by the original SOP) and the simplified expression:

Circuit Implementation

Implementation is Optional, for Extra Credit

Your steps from here are as follows:

1. Using switches for input, build and manually check a circuit for the S output bit of the full adder. Debug if necessary.
2. By modifying the mapping of inputs to outputs, use the Sketch from the prior validation step to drive validation of your S bit full adder circuit. Debug if necessary.
3. Demonstrate your working circuit for your instructor or TA.