Tiffany Q. Liu
February 7, 2011
CSC 270
Lab #2

Lab #2: Introduction to Transistors, Verifying Boolean Expressions with Python, and Building Adders with Logic Gates

Introduction

In this lab, we get an introduction to working with a transistor circuit. We also learn how to use Python code to verify the validity of our Boolean expressions. And finally, we use AND, OR, NOT, XOR, and NAND-gate circuits to build a 2-bit and 3-bit adder.

Materials

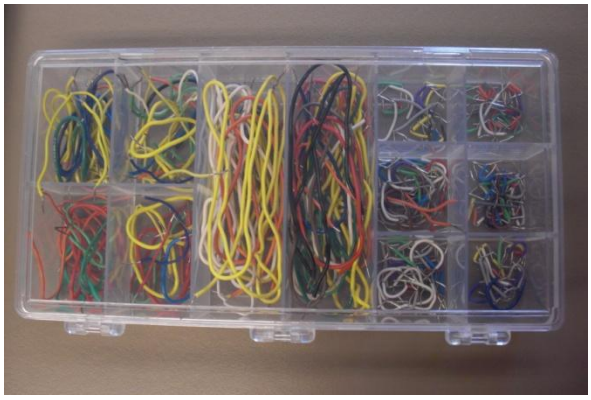

Figure 1. Wiring Kit.

Figure 2. Digital Training Kit.

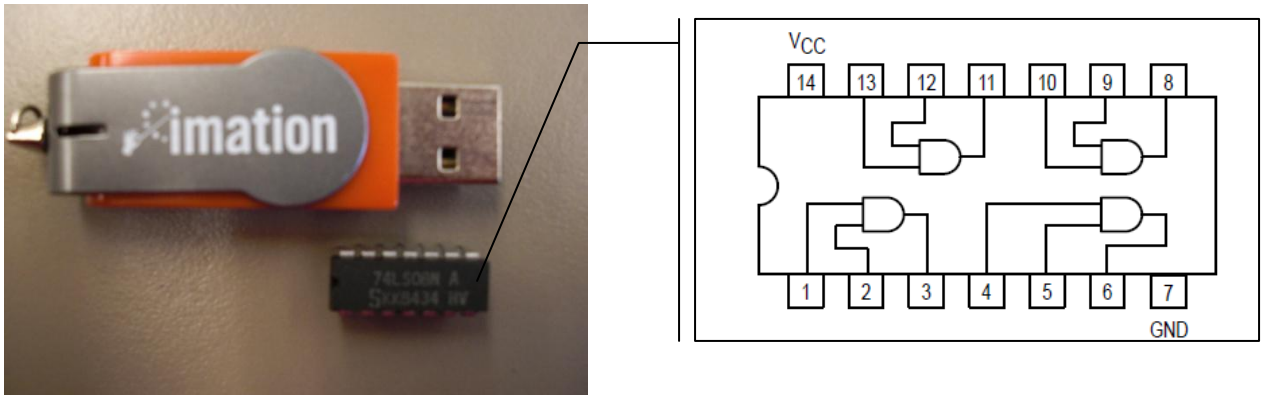

Figure 3. Quad 2-Input AND Gate 74LS08 Compared to a USB Flash Drive.

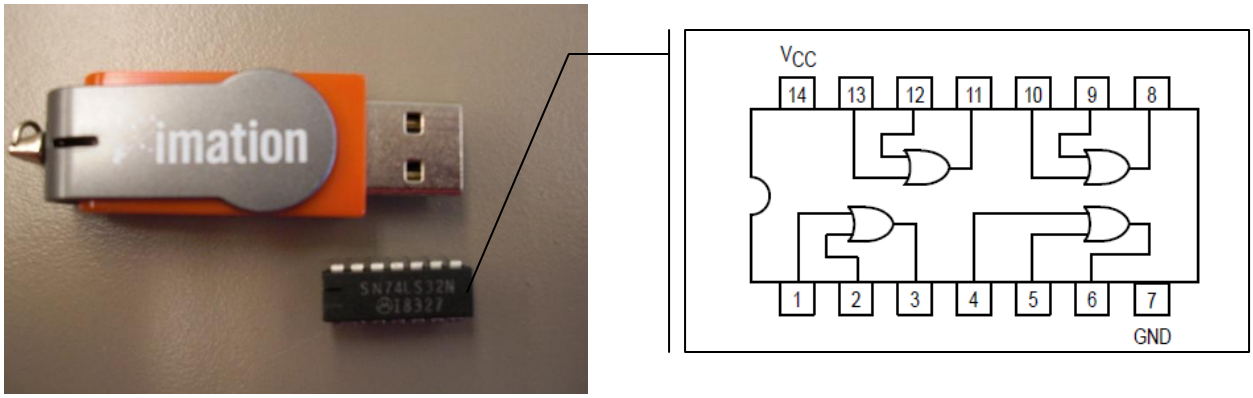

Figure 4. Quad 2-Input OR Gate 74LS32 Compared to a USB Flash Drive.

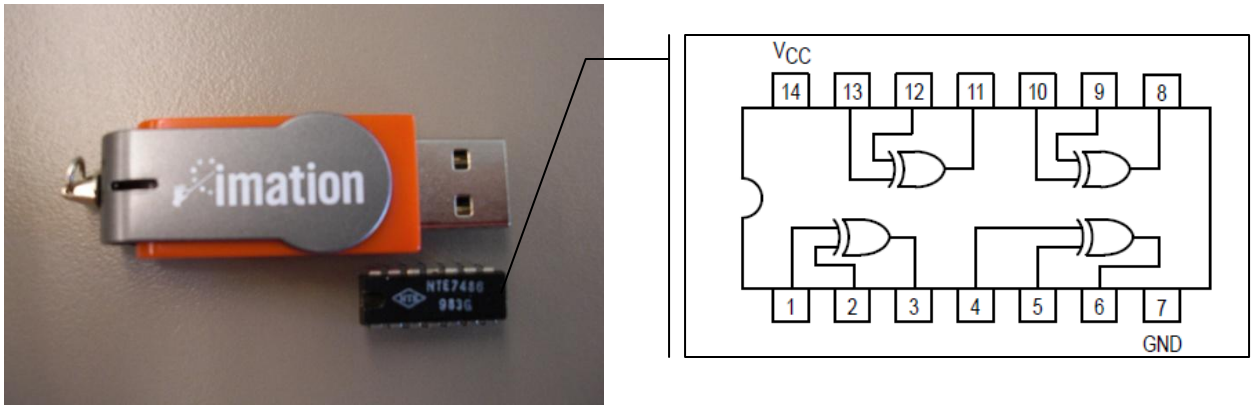

Figure 5. Quad 2-Input XOR Gate 74LS86 Compared to a USB Flash Drive.

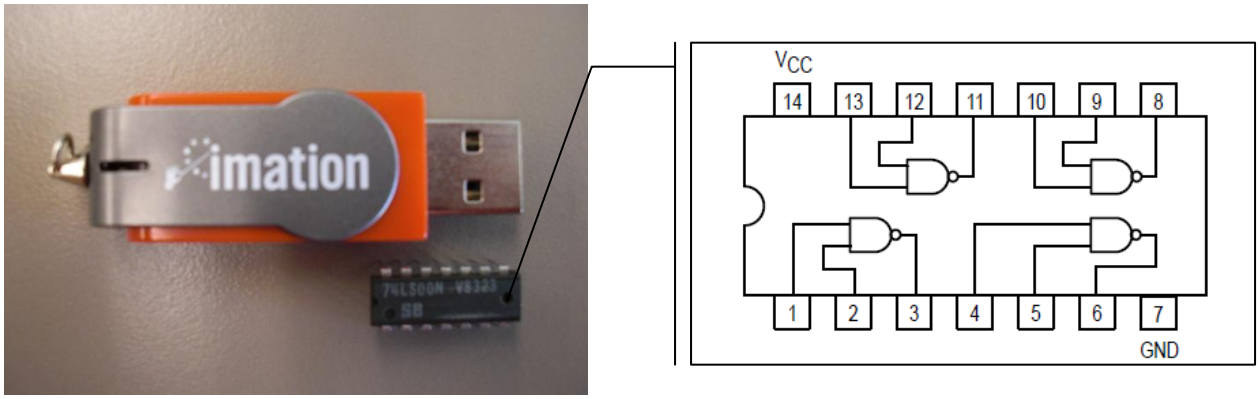

Figure 6. Quad 2-Input NAND Gate 74LS00 Compared to a USB Flash Drive.

The Transistor

The transistor is a three-pole semiconductor. The poles are polarized and have well-defined functions—the collector (C), the base (B), and the emitter (E). The transistor used in this lab is the 2N5772, which has the poles marked on the package. The following circuit (Fig.7) was built using the Digital Kit, wires, a transistor, a 1KΩ resistor, and a 10KΩ resistor. An interactive color-chart that can be found at: http://www.smpspowersupply.com/resistor/resistorcolorcode.html was used to double check the value of the resistors.

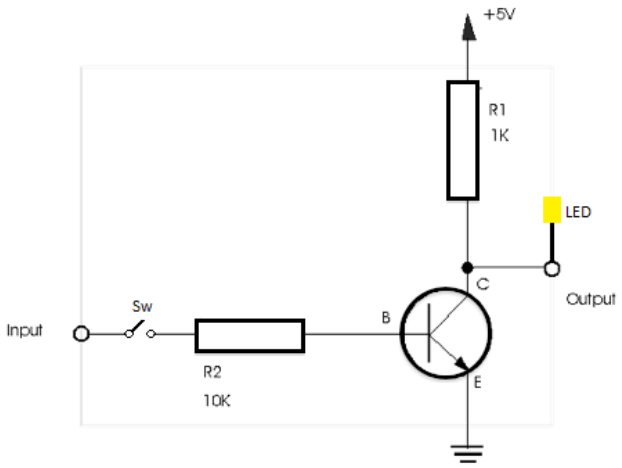

Figure 7. Transistor Circuit Diagram.

Once the circuit was built, the following truth table can be generated for it:

Input (Sw)	Output (LED)
OFF	ON
ON	OFF

Table 1. Truth Table for the Above Circuit (Fig. 7).

It can be seen that this truth table matches the truth table for an inverter (or NOT gate), where the 0's can represent OFF and 1's represent ON:

a	NOT a
0	1
1	0

Table 2. Truth Table for NOT Gate.

Thus, the transistor circuit (Fig. 7) implements the inverter (or NOT gate).

Verifying with Python

Running the following program, we verified that the output matched the truth table shown below:

```
# truthtable.py
# D. Thiebaut
# how a simple python program can generate the
# truth table of a boolean function
#
# here f is a function of 3 variables
#
# f = a.b + c
#
# g = a + b + c

def f( a, b, c ):
    return ( a & (not b) ) | c

def g( a, b, c ):

    return (not a) | (not b) | (not c)

def main():
    print "  a  b  c  |  f  g  "
    print "-----------+--------"
    for a in [ 0, 1 ]:
        for b in [ 0, 1 ]:
            for c in [ 0, 1 ]:
                print "%3d%3d%3d  |%3d%3d" % \
                      ( a, b, c, f( a, b, c ), g( a, b, c ) )

main()
```

Figure 8. Python Program Used to Generate Truth Tables for Boolean Functions.

```
  a  b  c  |  f  g
-----------+--------
  0  0  0  |  0  1
  0  0  1  |  1  1
  0  1  0  |  0  1
  0  1  1  |  1  1
  1  0  0  |  1  1
  1  0  1  |  1  1
  1  1  0  |  0  1
  1  1  1  |  1  0
```

Figure 9. Output to the Previous Python Code (Fig. 8).

The program was then modified to output the truth table for a 2-bit adder. The following is a truth table for a 2-bit binary adder generated by hand:

a	b	c	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Table 3. Truth Table for a 2-Bit Adder.

Using the modified Python program (Fig. 10), the below truth table was generated as the output (Fig. 11).

```
# truthTable.py
# D. Thiebaut
# how a simple python program can generate the
# truth table of a boolean function
#
# Modified by T. Liu and M. Walsh to output truth table for a 2-bit adder where:
#    sum (s) = a'.b + a.b' = a XOR b
#    carry (c) = a.b

# ----------------------------------------
# s(a,b) = a'.b + a.b'
#        = a XOR b
# ----------------------------------------
def s( a, b ):
    #return ( (not a) & b ) | ( a & (not b) )
    return a ^ b
# ----------------------------------------
# c(a,b) = a.b
# ----------------------------------------
def c( a, b ):
    return a and b
# ----------------------------------------
# Print out truth table for 2-bit adder.
# ----------------------------------------
def main():
    print "  a  b  |  c  s  "
    print "--------+--------"
    for a in [ 0, 1 ]:
        for b in [ 0, 1 ]:
            print "%3d%3d  |%3d%3d" % \
                    ( a, b, c( a, b ), s( a, b ) )

main()
```

Figure 10. Modified Python Program to Check Validity of 2-Bit Adder Truth Table.

```
  a  b  |  c  s
--------+--------
  0  0  |  0  0
  0  1  |  0  1
  1  0  |  0  1
  1  1  |  1  0
```

Figure 11. Output to Modified Python Code (Fig. 10).

From this, it can be seen that the hand-generated truth table for a 2-bit adder was verified by the Python code.

2-Bit Adder

From the truth table for a 2-bit binary adder (Table 3), a circuit can be built to generate the sum and the carry. Before building the circuit, we need to simplify the Boolean expressions for the sum (s) and carry (c) as much as possible to make the circuit as simple as possible:

$$s = a' \cdot b + a \cdot b'$$
$$= a \oplus b$$

$$c = a \cdot b$$

Using these Boolean expressions for s and c, the following circuit diagram was generated and verified against Table 3:

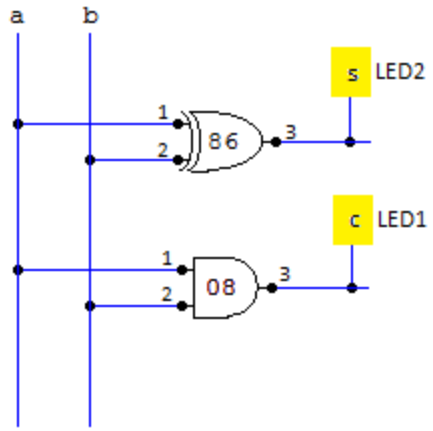

Figure 12. Circuit Diagram for a 2-Bit Adder.

The LED that indicated the s bit lit up only when either a or b was switched on, and the LED that indicated the c bit lit up only when both a and b were switched on, thus verifying that we successfully built a 2-bit adder using our logic gates.

3-Bit Adder

The following is a truth table for a 3-bit adder:

b_2	b_1	b_0	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Table 4. Truth Table for a 3-Bit Adder.

Using the above truth table (Table 4), a circuit can be built to generate the sum and the carry for a 3-bit adder. Again, before we build this circuit, we need to simplify our Boolean expressions for s and c so that we can make the circuit as simple as possible.

$$\begin{aligned} s &= b_2{}' \cdot b_1{}' \cdot b_0 + b_2{}' \cdot b_1 \cdot b_0{}' + b_2 \cdot b_1{}' \cdot b_0{}' + b_2 \cdot b_1 \cdot b_0 \\ &= b_2{}' \cdot (b_1{}' \cdot b_0 + b_1 \cdot b_0{}') + b_2 \cdot (b_1{}' \cdot b_0{}' + b_1 \cdot b_0) \\ &= b_2{}' \cdot (b_1 \oplus b_0) + b_2 \cdot (b_1 \oplus b_0)' \\ &= b_2 \oplus (b_1 \oplus b_0) \end{aligned}$$

$$\begin{aligned} c &= b_2{}' \cdot b_1 \cdot b_0 + b_2 \cdot b_1{}' \cdot b_0 + b_2 \cdot b_1 \cdot b_0{}' + b_2 \cdot b_1 \cdot b_0 \\ &= b_0 \cdot (b_2{}' \cdot b_1 + b_2 \cdot b_1{}') + (b_2 \cdot b_1) \cdot (b_0{}' + b_0) \\ &= b_0 \cdot (b_2 \oplus b_1) + (b_2 \cdot b_1) \end{aligned}$$

Using these Boolean expressions for s and c, the following circuit diagram was generated and verified against Table 4:

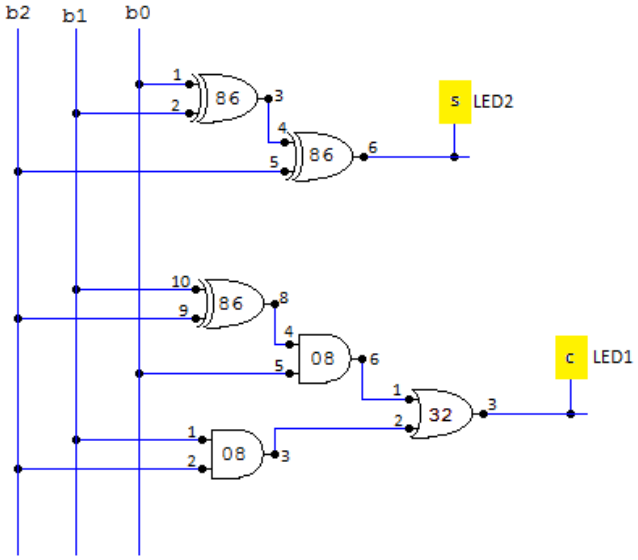

Figure 13. Circuit Diagram for a 3-Bit Adder.

The LED that indicated the s bit lit up only when one of b_2, b_1, or b_0 was switched on, the LED that indicated the c bit lit up only when two out of three of b_2, b_1, or b_0 were switched on, and both LEDs lit up only when all three of b_2, b_1, or b_0 were switched on, thus verifying that we successfully built a 2-bit adder using our logic gates.

2-Bit Adder with NAND Gates

Finally, we revisited the 2-bit adder circuit and worked with the Boolean expressions to only use NAND gates:

$$s = a' \cdot b + a \cdot b'$$
$$= (a' \cdot b)'' + (a \cdot b')''$$
$$Let\ K = (a' \cdot b)'\ and\ M = (a \cdot b')'$$
$$K = a' \uparrow b\ and\ M = a \uparrow b'$$
$$\therefore s = K' + M' = (K \cdot M)' = K \uparrow M$$
$$\therefore s = a' \uparrow b \uparrow a \uparrow b'$$
$$= (a \uparrow a) \uparrow b \uparrow a \uparrow (b \uparrow b)$$

$$c = a \cdot b$$
$$= (a \uparrow b) \uparrow (a \uparrow b)$$

Using these Boolean expressions for s and c, the following circuit diagram was generated:

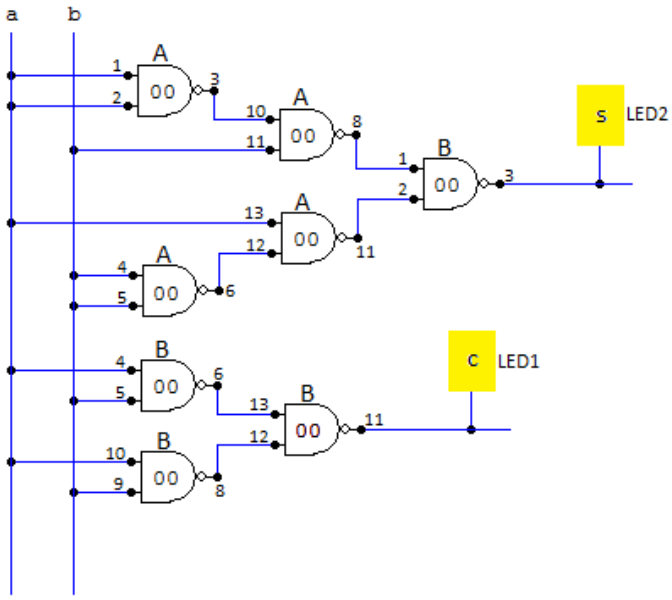

Figure 14. Circuit Diagram for a 2-Bit Adder Built Using Only NAND Gates.

This circuit worked just the same as the circuit built for the 2-bit adder in the previous section of the lab (Fig. 12).