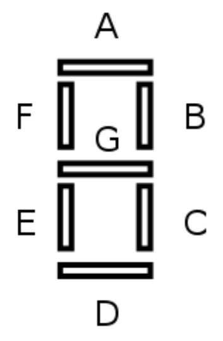
Number of Segments Circuit

Terry Sturtevant

Department of Physics and Computer Science, Wilfrid Laurier University

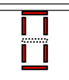
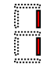
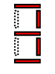
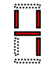
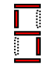
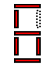
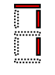
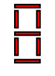
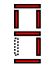
Introduction

Seven segment displays are common in electronics. They consist of an arrangement of LEDS as shown below:



The segments are identified as *a,b,c,d,e,f,g*.

The table below shows how numbers are typically displayed and how many segments are lit in each case:

number	display	segments lit	number of segments lit
0		a,b,c,d,e,f	6
1		b,c	2
2		a,b,d,e,g	5
3		a,b,c,d,g	5
4		b,c,f,g	4
5		a,c,d,f,g	5
6		a,c,d,e,f,g	6
7		a,b,c	3
8		a,b,c,d,e,f,g	7
9		a,b,c,d,f,g	6

The **Number of Segments Circuit** indicates how many segments will be lit in a 7-segment display for a number from 0 to 9.

Logic Design

For this particular problem, the truth table looks like this:

number	$a_3a_2a_1a_0$	number of segments	$b_2b_1b_0$
0	0000	6	110
1	0001	2	010
2	0010	5	101
3	0011	5	101
4	0100	4	100
5	0101	5	101
6	0110	6	110
7	0111	3	011
8	1000	7	111
9	1001	6	110
10	1010	error	000
11	1011	error	000
12	1100	error	000
13	1101	error	000
14	1110	error	000
15	1111	error	000

In order to determine the logic equations for the *Number of Segments Circuit*, the only quantities to consider are the binary inputs and outputs. **The fact that the bits are grouped together to represent numbers is irrelevant.**
The truth table containing only binary quantities is shown below:

a_3	a_2	a_1	a_0	b_2	b_1	b_0
0	0	0	0	1	1	0
0	0	0	1	0	1	0
0	0	1	0	1	0	1
0	0	1	1	1	0	1
0	1	0	0	1	0	0
0	1	0	1	1	0	1
0	1	1	0	1	1	0
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	0	1	1	1	0
1	0	1	0	0	0	0
1	0	1	1	0	0	0
1	1	0	0	0	0	0
1	1	0	1	0	0	0
1	1	1	0	0	0	0
1	1	1	1	0	0	0

Simplifying Equations

Here is the section of the truth table for output b_0.

a_3	a_2	a_1	a_0	b_0
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

A Karnaugh map was used to determine simplified sum-of-products logic equations for each output.

This produces the following Karnaugh map for bit b_0:

		a_1a_0			
b_0		00	01	11	10
a_3a_2	00	0	0	1	1
	01	0	1	1	0
	11	0	0	0	0
	10	1	0	0	0

Groups of 1's can be identified.

		a_1a_0			
b_0		00	01	11	10
a_3a_2	00	0	0	1	1
	01	0	1	1	0
	11	0	0	0	0
	10	1	0	0	0

This will produce an equation for b_0.

$$b_0 = \overline{a_3}\ \overline{a_2}\ a_1 + \overline{a_3}\ a_2\ a_0 + a_3\overline{a_2}\ \overline{a_1}\ \overline{a_0}$$

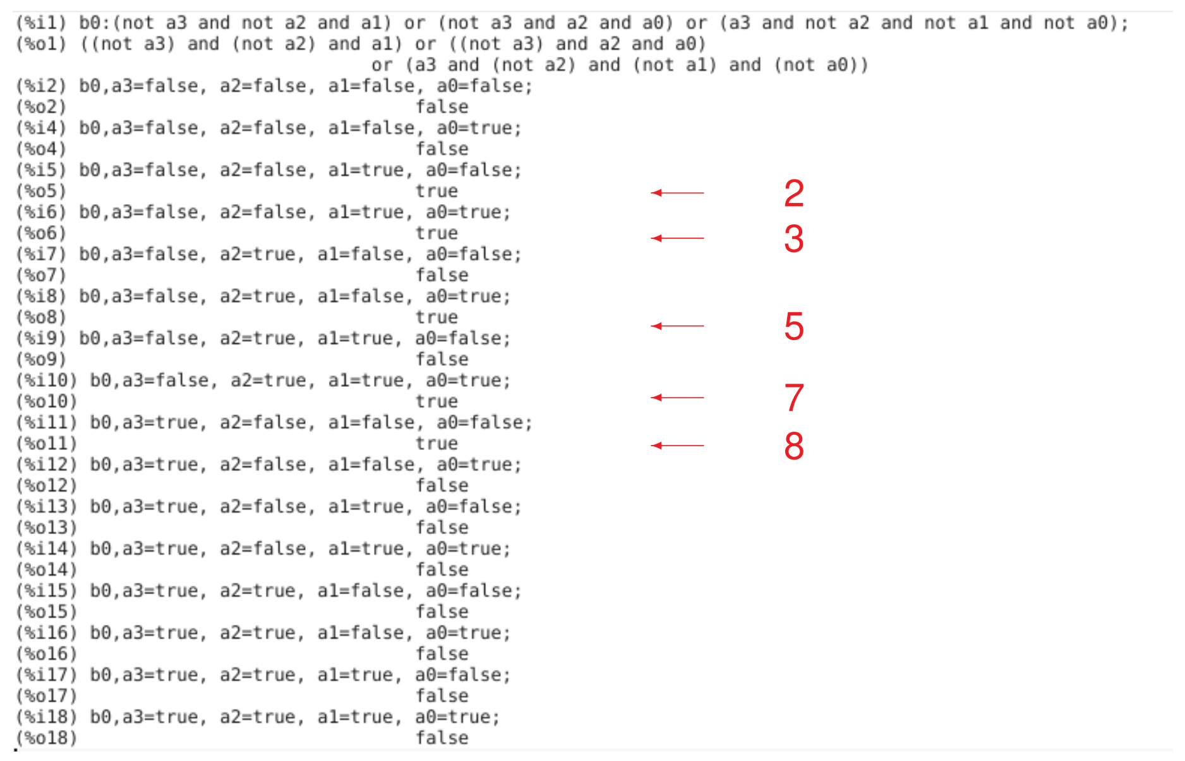
A similar process provides the equations for the other outputs. The equations for the other outputs are:

$$b_1 = \overline{a_2}\ \overline{a_1} + \overline{a_3}\ a_2\ a_1$$

$$b_2 = \overline{a_3}\ \overline{a_0} + a_3\ \overline{a_2}\ \overline{a_1} + \overline{a_3}\ \overline{a_2}\ a_1 + \overline{a_3}\ a_2\ \overline{a_1}$$

Testing Equations

Maxima was used to test the equations.
Since b_0 is the least significant bit, it will be high whenever an *odd* number of segments are lit.

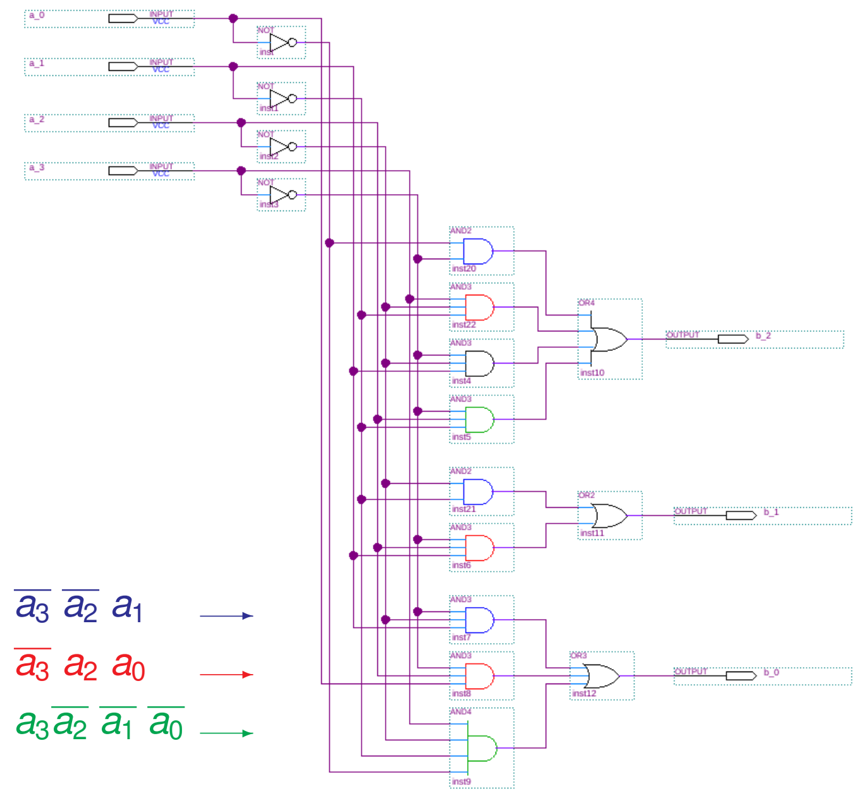


```
(%i1) b0:(not a3 and not a2 and a1) or (not a3 and a2 and a0) or (a3 and not a2 and not a1 and not a0);
(%o1) ((not a3) and (not a2) and a1) or ((not a3) and a2 and a0)
                                   or (a3 and (not a2) and (not a1) and (not a0))
(%i2) b0,a3=false, a2=false, a1=false, a0=false;
(%o2)                               false
(%i4) b0,a3=false, a2=false, a1=false, a0=true;
(%o4)                               false
(%i5) b0,a3=false, a2=false, a1=true, a0=false;
(%o5)                               true
(%i6) b0,a3=false, a2=false, a1=true, a0=true;
(%o6)                               true
(%i7) b0,a3=false, a2=true, a1=false, a0=false;
(%o7)                               false
(%i8) b0,a3=false, a2=true, a1=false, a0=true;
(%o8)                               true
(%i9) b0,a3=false, a2=true, a1=true, a0=false;
(%o9)                               false
(%i10) b0,a3=false, a2=true, a1=true, a0=true;
(%o10)                              true
(%i11) b0,a3=true, a2=false, a1=false, a0=false;
(%o11)                              true
(%i12) b0,a3=true, a2=false, a1=false, a0=true;
(%o12)                              false
(%i13) b0,a3=true, a2=false, a1=true, a0=false;
(%o13)                              false
(%i14) b0,a3=true, a2=false, a1=true, a0=true;
(%o14)                              false
(%i15) b0,a3=true, a2=true, a1=false, a0=false;
(%o15)                              false
(%i16) b0,a3=true, a2=true, a1=false, a0=true;
(%o16)                              false
(%i17) b0,a3=true, a2=true, a1=true, a0=false;
(%o17)                              false
(%i18) b0,a3=true, a2=true, a1=true, a0=true;
(%o18)                              false
```

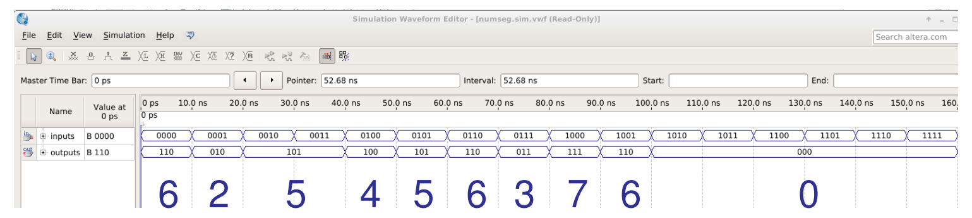
An odd number of segments are used for the numbers 2,3, 5, 7, and 8. These were the only cases where b_0 was true.
This verifies that the equation is correct.
The same process was used to verify that the other equations were also correct.

Circuit Drawing and Simulation

The circuit looks like this, with the terms in b_0 highlighted:


$\overline{a_3}\ \overline{a_2}\ a_1$
$\overline{a_3}\ a_2\ a_0$
$a_3\overline{a_2}\ \overline{a_1}\ \overline{a_0}$

The simulation output looks like this:



Simulation Waveform Editor - [numseg.sim.vwf (Read-Only)]

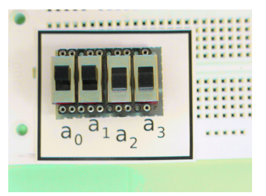
Master Time Bar: 0 ps | Pointer: 52.68 ns | Interval: 52.68 ns | Start: | End:

```
Name      Value at   0 ps  10.0 ns  20.0 ns  30.0 ns  40.0 ns  50.0 ns  60.0 ns  70.0 ns  80.0 ns  90.0 ns  100.0 ns  110.0 ns  120.0 ns  130.0 ns  140.0 ns  150.0 ns  160.
          0 ps       0 ps
inputs    B 0000     0000  0001  0010  0011  0100  0101  0110  0111  1000  1001  1010  1011  1100  1101  1110  1111
outputs   B 110      110   010   101         100   101   110   011   111   110   000
```

This verifies that circuit correctly implements the equations.

Testing

This section will vary a lot by project; you want to make it easy for people to test the circuit to see that it performs as expected.



It may inlcude images, and/or refer to earlier things on the poster.