Taylor Cowley
HW5
May 14 2016

ECEn 220
Chapter 9 Homework

1. Convert the following to **unsigned binary**, add the numbers together in binary, and check your result by converting it back to base-10.

   3 + 7 =? Use four bits for your operands and result.

   0011 + 0111 = 1010 = 10

   9 + 4 =? Use four bits for your operands and result.

   1001 + 0100 = 1101 = 13

   13 + 12 =? Use five bits for your operands and result.

   01101 + 1100 = 11001 = 25

   01100

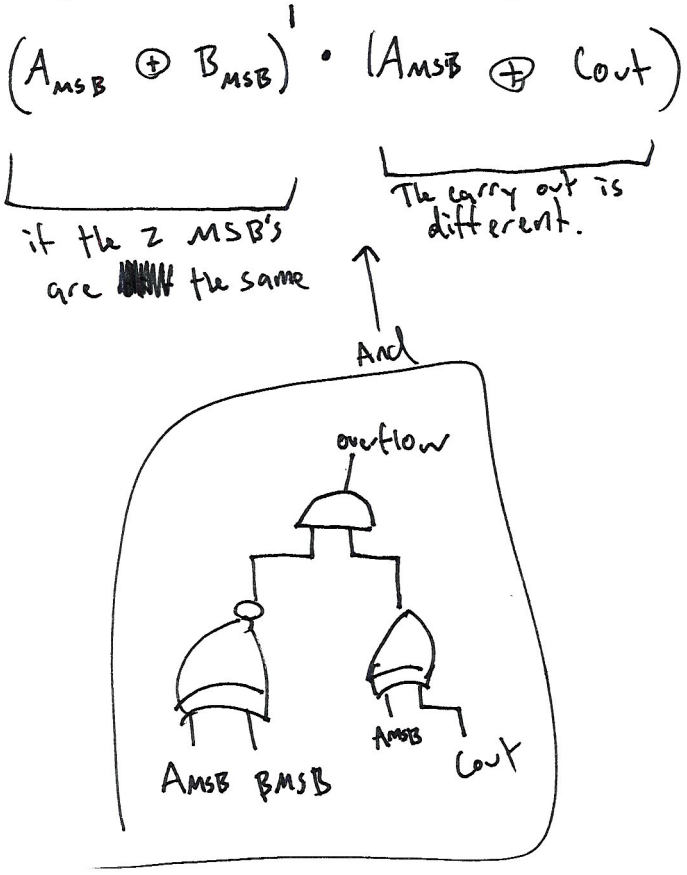
2. Convert the following to **2's complement**, add the numbers together in binary. In each case first do the addition and compute a result with the same number of bits as the operands. Then, determine which operations overflow. For those overflow, first sign-extend the operands by one bit position and repeat the operations. Check your result by converting it back to base-10

   3 + 7=? Use 4 bits for operands and result. Sign-extend them to 5 bits and do it again if overflow is detected.

   0011 3
   + 0111 7
   1010 = -6
   overflow

   00011 3
   + 00111 7
   01010 = 10

5. The chapter discussed detecting overflow in 2's complement addition by looking at the sign of both operands and the sign of the result. Derive the logic equation required to detect overflow this way and draw a gate-level schematic of the resulting overflow detection circuit.



ECEn 220
Appendix A Homework

A.1 Implement a 4:1 MUX using built-in gates. Draw the schematic of the corresponding circuit.

A.2. Implement a 2:4 decoder using built-in gates. Draw the schematic of the corresponding circuit.

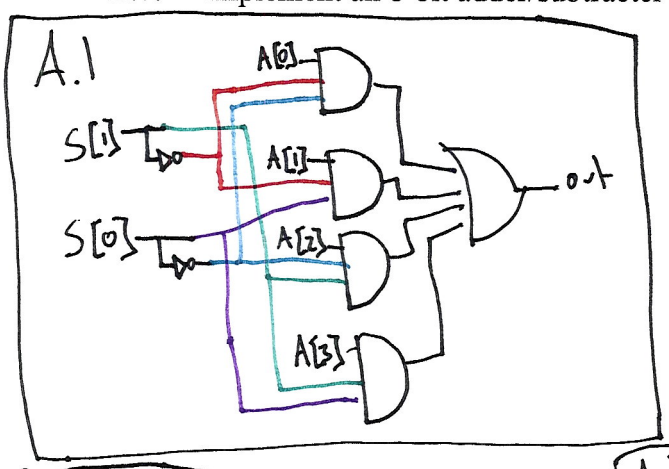
A.3. Implement a 4:1 MUX using your 2:4 decoder and some built-in gates. Draw the schematic of the corresponding circuit. You can show the 2:4 decoder as a block in this schematic (no need to break it down to its constituent gates).

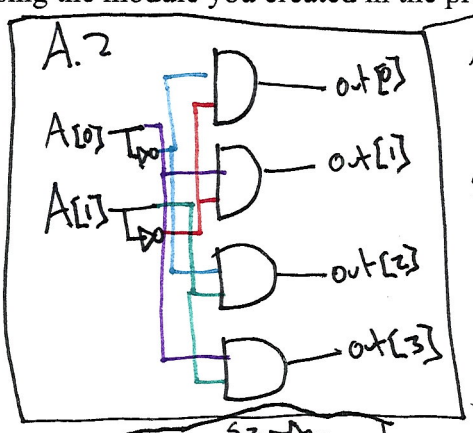
A.4. Implement a 3-input XOR gate using *not*, *and*, and *or* gates. Draw the schematic of the corresponding circuit.

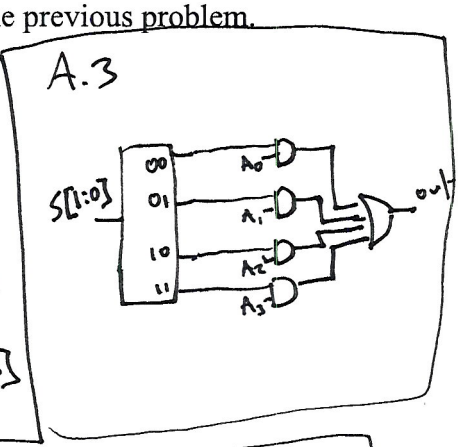
A.5. Implement a 3:8 decoder using two 2:4 decoders and some gates. Draw the schematic of the corresponding circuit.

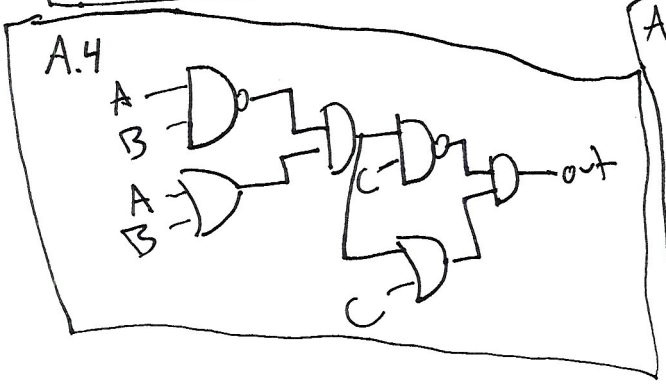
A.6. Implement a 1-bit adder/subtracter module using built-in gates. Use the method of Section 8.7 to construct it. —This doesn't have any meaning. I'm assuming 9.7

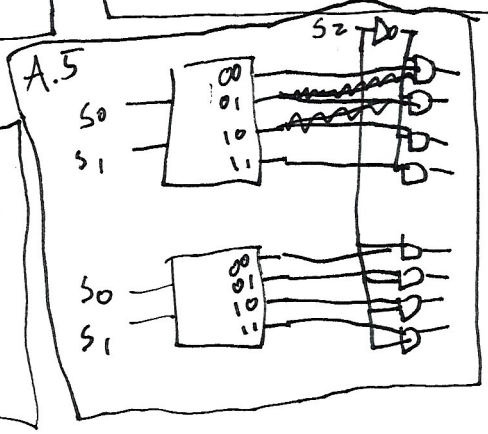
A.7. Implement an 8-bit adder/subtracter using the module you created in the previous problem.











A.6, A.7
no drawing necessary

```
1   // Taylor Cowley
2   // Hw5 Appendix A problems
3   // They were horrible.
4
5   module mux_4_1(out, a, select);
6      input[3:0] a;
7      input[1:0] select;
8      output out;
9
10     wire s1_not, s0_not, and0, and1, and2, and3;
11
12     not(s1_not, select[1]);
13     not(s0_not, select[0]);
14
15     and(and0, s1_not, s0_not, a[0]);
16     and(and1, s1_not, select[0], a[1]);
17     and(and2, select[1], s0_not, a[2]);
18     and(and3, select[1], select[0], a[3]);
19     or(out, and0, and1, and2, and3);
20  endmodule
21
22
23  module decoder_2_4(out, a);
24     input[1:0] a;
25     output[3:0] out;
26
27     wire a1_not, a0_not;
28     not(a1_not, a[1]);
29     not(a0_not, a[0]);
30
31     and(out[0], a1_not, a0_not);
32     and(out[1], a1_not, a[0]);
33     and(out[2], a[1], a0_not);
34     and(out[3], a[1], a[0]);
35  endmodule
36
37
38  module mux_4_1_using_decoder(out, a, select);
39     input[3:0] a;
40     input[1:0] select;
41     output out;
42
43     wire[3:0] decoder_output;
44     wire and0,and1,and2,and3;
45
46     decoder_2_4 the_thing(decoder_output[3:0], select);
47     and(and0, a[0], decoder_output[0]);
48     and(and1, a[1], decoder_output[1]);
49     and(and2, a[2], decoder_output[2]);
50     and(and3, a[3], decoder_output[3]);
51     or(out, and0, and1, and2, and3);
52  endmodule
53
54
55  module xor_using_gates(out, a, b, c);
56     input a;
```

```
57      input b;
58      input c;
59      output out;
60
61      // I am tired of defining wires. They are not mandatory in verilog
62      nand(n_out, a, b);
63      or(o_out, a, b);
64      and(a_out, n_out, o_out);
65      nand(n_out2, a_out, c);
66      or(o_out2, a_out, c);
67      and(out, n_out2, o_out2);
68  endmodule
69
70
71  module mega_decoder(out, select);
72      input[2:0] select;
73      output[7:0] out;
74      wire[3:0] d1_out, d2_out;
75      decoder_2_4 d1(d1_out[3:0], select[1:0]);
76      decoder_2_4 d2(d2_out[3:0], select[1:0]);
77
78      not(s_2_not, select[2]);
79      and(out[0], d1_out[0], s_2_not);
80      and(out[1], d1_out[1], s_2_not);
81      and(out[2], d1_out[2], s_2_not);
82      and(out[3], d1_out[3], s_2_not);
83
84      and(out[4], d2_out[0], select[2]);
85      and(out[5], d2_out[1], select[2]);
86      and(out[6], d2_out[2], select[2]);
87      and(out[7], d2_out[3], select[2]);
88  endmodule
89
90  //Required to implement the subtractor of 9.7
91  module full_adder(sum, cout, a, b, cin);
92      // Ports
93      input a;
94      input b;
95      input cin;
96      output cout;
97      output sum;
98
99      wire a_b, b_cin, a_cin;
100
101     and(a_b, a, b);
102     and(b_cin, b, cin);
103     and(a_cin, a, cin);
104     or(cout, a_b, b_cin, a_cin);
105
106     xor(sum, a, b, cin);
107 endmodule
108
109 module add_sub(sum, cout, a, b, sub);
110     // Ports
111     input a;
112     input b;
```

```
113      input sub;
114      output cout;
115      output sum;
116
117      wire sub_not;
118      not(sub_not, sub);
119
120      xor(b_x, b, sub_not);
121      full_adder full_add(sum, cout, a, b_x, 0);
122   endmodule
123
124
125   module add_sub_8(sum, cout, a, b, sub);
126      input[7:0] a;
127      input[7:0] b;
128      input sub;
129      output cout;
130      output[7:0] sum;
131
132      not(s_not, sub);
133
134      xor(b0, b[0], s_not);
135      xor(b1, b[1], s_not);
136      xor(b2, b[2], s_not);
137      xor(b3, b[3], s_not);
138      xor(b4, b[4], s_not);
139      xor(b5, b[5], s_not);
140      xor(b6, b[6], s_not);
141      xor(b7, b[7], s_not);
142
143      full_adder add1(sum[0], cout0, a[0], b0, 0);
144      full_adder add2(sum[1], cout1, a[1], b1, cout0);
145      full_adder add3(sum[2], cout2, a[2], b2, cout1);
146      full_adder add4(sum[3], cout3, a[3], b3, cout2);
147      full_adder add5(sum[4], cout4, a[4], b4, cout3);
148      full_adder add6(sum[5], cout5, a[5], b5, cout4);
149      full_adder add7(sum[6], cout6, a[6], b6, cout5);
150      full_adder add8(sum[7], cout, a[7], b7, cout6);
151   endmodule
152
```