Problem 1. In class we designed a binary adder by starting with a truth-table definition for a bit-slice and extending this in a manner that resembled traditional addition in positional notation. Design a subtractor in this same fashion. Develop the truth table and gate-level circuit diagram for a bit slice. Show how these are assembled to form an 8-bit subtractor. How does this compare to the “negate and add” design that we did in class?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>P</th>
<th>Difference</th>
<th>Borrow</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{DIFFERENCE} = A \oplus B \oplus P
\]

\[
\text{BORROW} = B \cdot P + A' \cdot (B + P)
\]
Problem 2. Show how to configure the xilinx CLB below to form a 2-bit adder slice.

Problem 3. Complete the following math worksheet to familiarize yourself with arithmetic dealing with different number representations as well as different bases. For all problems, assume the maximum number of digits for the solution is 4. Note any overflow that will result in an answer over 4 digits, then write what
the answer would be if there was no constraint on the number of digits. Work by hand, and show your work!

<table>
<thead>
<tr>
<th>Base 2 (Sign Magnitude)</th>
<th>1101 +0110 0011 1011 -0001 1100</th>
<th>0010 +0011 0101 1100</th>
<th>1011 +1001 1100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1101 +0110 0011 1011 -0001 1100</td>
<td>0010 +0011 0101 1100</td>
<td>1011 +1001 1100</td>
</tr>
<tr>
<td>Base 2 (Two's Complement)</td>
<td>1101 +0110 0011 1011 -0001 1100</td>
<td>0010 +0011 0101 1100</td>
<td>1011 =&gt; 11011 +1001 11001 (overflow) 10100</td>
</tr>
<tr>
<td></td>
<td>1101 +0110 0011 1011 -0001 1100</td>
<td>0010 +0011 0101 1100</td>
<td>1011 =&gt; 00101 -1010 -11010 (overflow) 01011</td>
</tr>
<tr>
<td>Base 10 (digits are 0-9)</td>
<td>1337 +1337 2674</td>
<td>5537 +8008 13545 (overflow)</td>
<td>1234 -2345 -1111</td>
</tr>
<tr>
<td>Base 7 (digits are 0-6)</td>
<td>1234 +3601 5135 -6006 +1111 -4565 -5432 -1036 -6501</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base 13 (digits are 0-C)</td>
<td>ABC0 +12 34 C124 CAB7 -BAC2 0CC5 CCCC +AAAA 1AAA9 (overflow)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base 16 (Hex) (digits are 0-F)</td>
<td>BADF +23CE DEAD</td>
<td>9999 +5432 EDCB</td>
<td>DADA -BEED 1BED</td>
</tr>
</tbody>
</table>
Problem 4. Design a controller for 16-bit version of the serial unsigned multiplier that we developed in class. It has two 16-bit inputs A and B, a start signal, and a select control signal HILO that determines whether the upper (1) or lower (0) half of the result is accessible on the 16-bit output. It should also have a control output VALID indicating that the output is valid. It also has a clock input. Valid data inputs are provided and the start signal is raised to start the multiplier on the rising edge. It runs on its own and eventually asserts VALID. Outputs remain available until the next operation is started.

Describe how you would modify your design to use a partial combinational multiplier capable of multiplying the 16 bit multiplicand by a 4-bit multiplier?
Problem 5. In this problem you are given a verilog full adder module with simulation delays. It is contained in the [http://inst.eecs.berkeley.edu/~cs150/fa07/HWs/full_adder.v](http://inst.eecs.berkeley.edu/~cs150/fa07/HWs/full_adder.v)

a. First simulate this module adding 0+0, 1+0, and 1+1 and turn in timing simulation waveform from Modelsim showing the propagation delays.

b. Next using the fuller adder module construct a 32-bit ripple carry adder. Simulate the design adding two numbers that will show the worst case delay. What numbers did you use? 0xFFFFFFFF + 1

Why will they give the worst case delay? Because the carry has to propagate down the whole system.

Turn in the verilog and the timing simulation.

```verilog
module rca(sum,cout,a,b,cin);
    input [31:0] a,b;
    input cin;
    output [31:0] sum;
    output cout;
    wire [31:1] c;

    full_adder FA0 (sum[0], c[1], a[0], b[0], cin);
    full_adder FA1 (sum[1], c[2], a[1], b[1], c[1]);
    full_adder FA2 (sum[2], c[3], a[2], b[2], c[2]);
    full_adder FA3 (sum[3], c[4], a[3], b[3], c[3]);
    full_adder FA4 (sum[4], c[5], a[4], b[4], c[4]);
    full_adder FA5 (sum[5], c[6], a[5], b[5], c[5]);
    full_adder FA6 (sum[6], c[7], a[6], b[6], c[6]);
    full_adder FA7 (sum[7], c[8], a[7], b[7], c[7]);
    full_adder FA8 (sum[8], c[9], a[8], b[8], c[8]);
    full_adder FA9 (sum[9], c[10], a[9], b[9], c[9]);
    full_adder FA10 (sum[10], c[11], a[10], b[10], c[10]);
    full_adder FA11 (sum[11], c[12], a[11], b[11], c[11]);
    full_adder FA12 (sum[12], c[13], a[12], b[12], c[12]);
    full_adder FA13 (sum[13], c[14], a[13], b[13], c[13]);
    full_adder FA14 (sum[14], c[15], a[14], b[14], c[14]);
    full_adder FA15 (sum[15], c[16], a[15], b[15], c[15]);
    full_adder FA16 (sum[16], c[17], a[16], b[16], c[16]);
    full_adder FA17 (sum[17], c[18], a[17], b[17], c[17]);
    full_adder FA18 (sum[18], c[19], a[18], b[18], c[18]);
    full_adder FA19 (sum[19], c[20], a[19], b[19], c[19]);
    full_adder FA20 (sum[20], c[21], a[20], b[20], c[20]);
    full_adder FA21 (sum[21], c[22], a[21], b[21], c[21]);
    full_adder FA22 (sum[22], c[23], a[22], b[22], c[22]);
    full_adder FA23 (sum[23], c[24], a[23], b[23], c[23]);
    full_adder FA24 (sum[24], c[25], a[24], b[24], c[24]);
    full_adder FA25 (sum[25], c[26], a[25], b[25], c[25]);
    full_adder FA26 (sum[26], c[27], a[26], b[26], c[26]);
```
full_adder FA27 (sum[27], c[28], a[27], b[27], c[27]);
full_adder FA28 (sum[28], c[29], a[28], b[28], c[28]);
full_adder FA29 (sum[29], c[30], a[29], b[29], c[29]);
full_adder FA30 (sum[30], c[31], a[30], b[30], c[30]);
full_adder FA31 (sum[31], cout, a[31], b[31], c[31]);

end module

c. Next using the fuller adder modules construct a 16-bit carry-select adder. Simulate the design adding two numbers that will show the worst case delay.
   What numbers did you use? 0xFFFF +1
d. Why will they give the worst case delay? Because the Carry has to propagate the longest path which 4 bits
e. Turn in the verilog and the timing simulation.

module csa(sum, cout, a, b, cin);
input [15:0] a, b;
input cin;
output [15:0] sum;
output cout;
wire c, c1, c2, c1_0, c1_1, c2_0, c2_1, c3_0, c3_1;
wire [3:0] s1_0, s1_1, s2_0, s2_1, s3_0, s3_1;

full_adder_4 fa0(sum[3:0], c, a[3:0], b[3:0], cin);
full_adder_4 fa1_0(s1_0[3:0], c1_0, a[7:4], b[7:4], 0);
full_adder_4 fa1_1(s1_1[3:0], c1_1, a[7:4], b[7:4], 1);
full_adder_4 fa2_0(s2_0[3:0], c2_0, a[11:8], b[11:8], 0);
full_adder_4 fa2_1(s2_1[3:0], c2_1, a[11:8], b[11:8], 1);
full_adder_4 fa3_0(s3_0[3:0], c3_0, a[15:12], b[15:12], 0);
full_adder_4 fa3_1(s3_1[3:0], c3_1, a[15:12], b[15:12], 1);

assign c1 = (c) ? c1_1 : c1_0;
assign c2 = (c1) ? c2_1 : c2_0;
assign cout = (c2) ? c3_1 : c3_0;
assign sum[7:4] = (c) ? s1_1[3:0] : s1_0[3:0];
assign sum[11:8] = (c1) ? s2_1[3:0] : s2_0[3:0];
assign sum[15:12] = (c2) ? s3_1[3:0] : s3_0[3:0];

endmodule

module full_adder_4(sum, cout, a, b, cin);
input [3:0] a, b;
input cin;
output [3:0] sum;
output cout;
wire [3:1] c;

full_adder FA0 (sum[0], c[1], a[0], b[0], cin);
full_adder FA1 (sum[1], c[2], a[1], b[1], c[1]);
full_adder FA2 (sum[2], c[3], a[2], b[2], c[2]);
full_adder FA3 (sum[3], cout, a[3], b[3], c[3]);

endmodule

f. How you can you improve the performance of the carry-select adder? Alter your verilog so that you get improved performance. Turn in a timing simulation for your new design adding the same numbers as you did for the carry-select adder.

You can make the ripple carry portion of the carry-select adder less bits. This will increase the area of the design but give you less propagation delay for the carry.