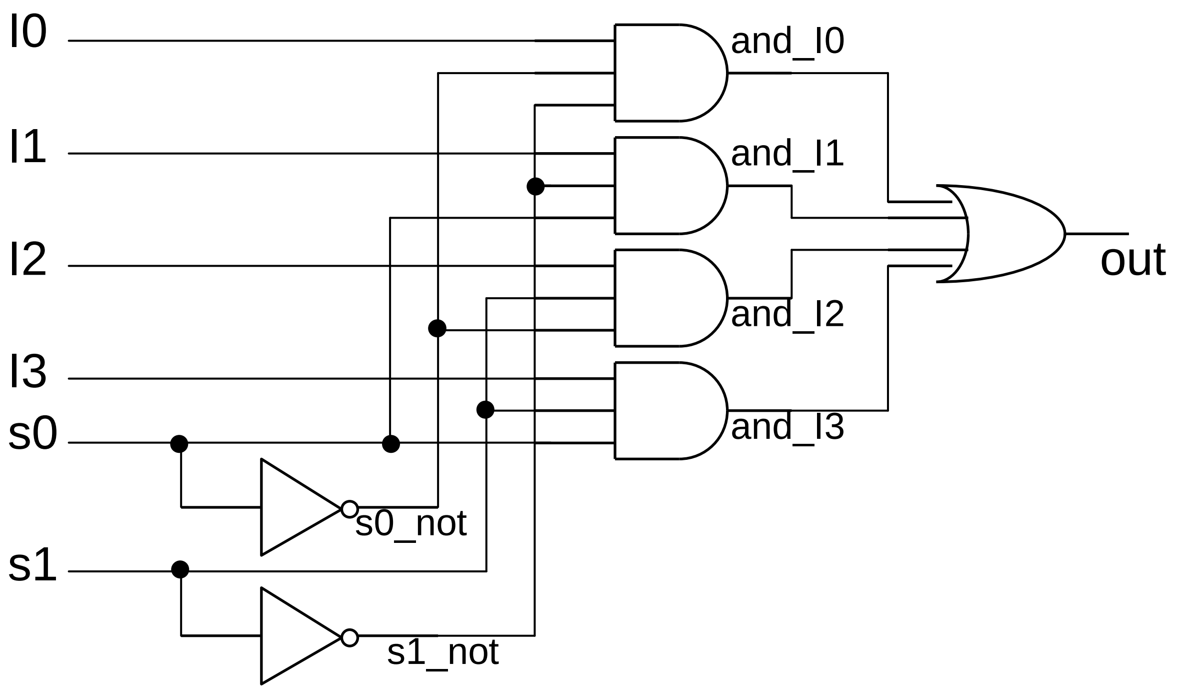
ECEn/CS 224
Appendix A Homework Solutions

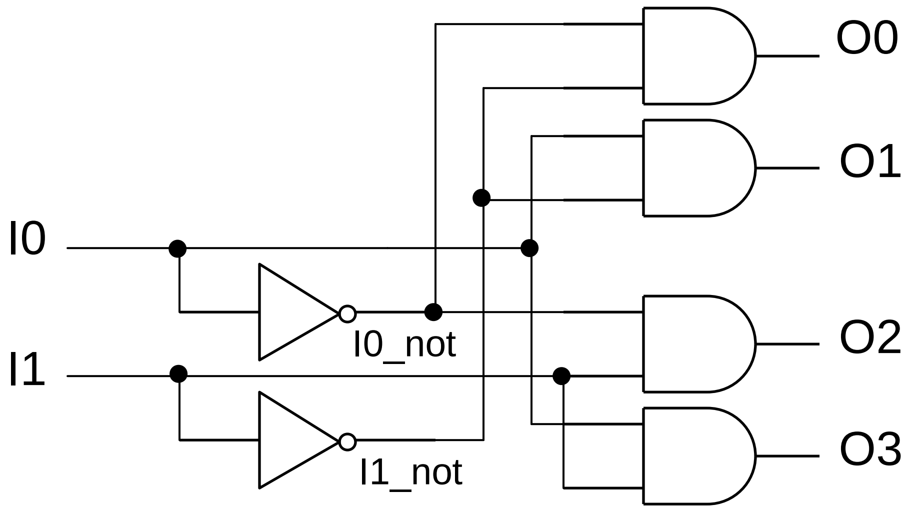
A.1 Implement a 4:1 MUX using built-in gates. Draw the schematic of the corresponding circuit.

```
module mux41(out, I0, I1, I2, I3, s0, s1);
   input I0, I1, I2, I3, s0, s1;
   output out;
   wire s0_not, s1_not, and_I0, and_I1, and_I2, and_I3;
   not(s0_not, s0);
   not(s1_not, s1);
   and(and_I0, I0, s0_not, s1_not);
   and(and_I1, I1, s0, s1_not);
   and(and_I2, I2, s0_not, s1);
   and(and_I3, I3, s0, s1);
   or(out, and_I0, and_I1, and_I2, and_I3);
endmodule
```



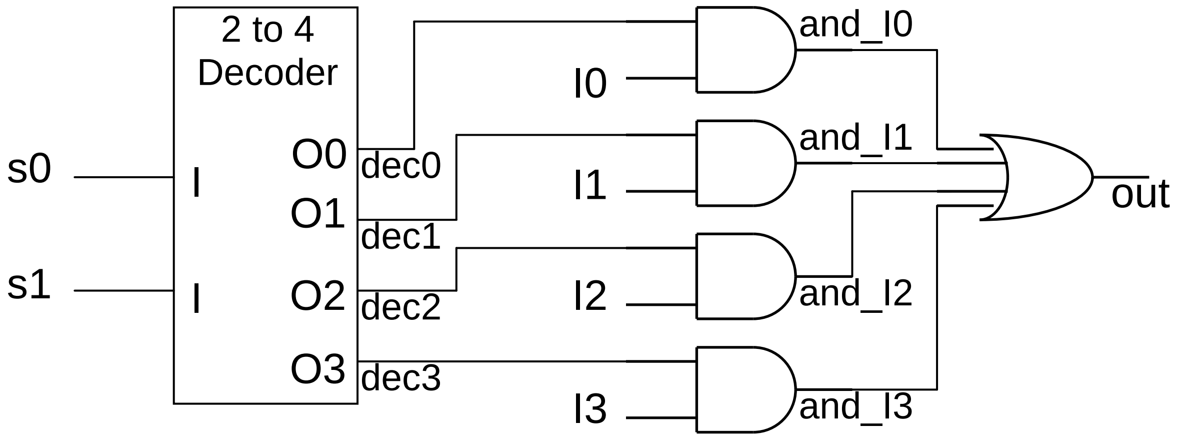
A.2. Implement a 2:4 decoder using built-in gates. Draw the schematic of the corresponding circuit.

```
module decoder24 (O0, O1, O2, O3, I0, I1);
   input I0, I1;
   output O0, O1, O2, O3;
   wire I0_not, I1_not;
   not(I0_not, I0);
   not(I1_not, I1);
   and(O0, I0_not, I1_not);
   and(O1, I0, I1_not);
   and(O2, I0_not, I1);
   and(O3, I0, I1);
endmodule
```



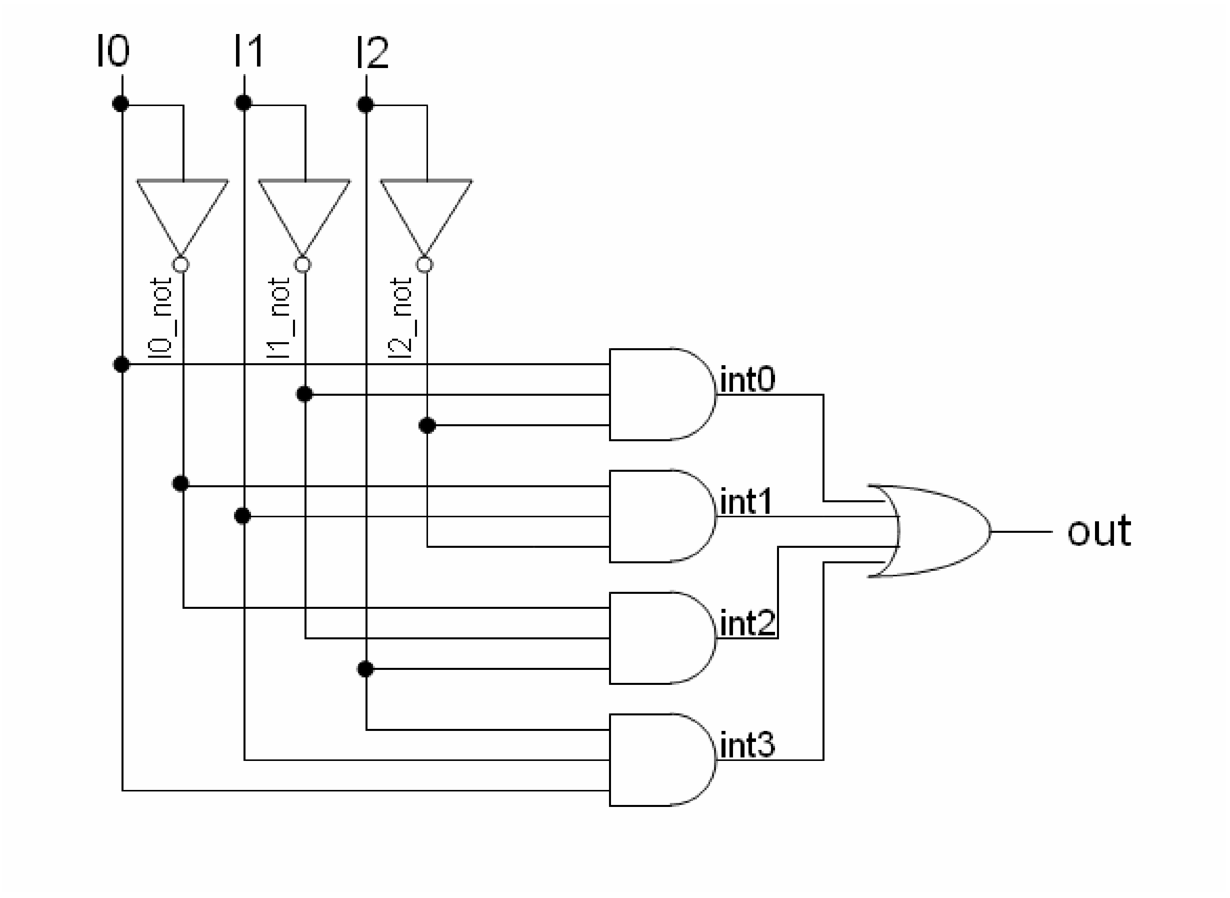
A.3. Implement a 4:1 MUX using your 2:4 decoder and some built-in gates. Draw the schematic of the corresponding circuit. You can show the 2:4 decoder as a block in this schematic (no need to break it down to its constituent gates).

```
module mux41_alt(out, I0, I1, I2, I3, s0, s1);
   input I0, I1, I2, I3, s0, s1;
   output out;
   wire dec0, dec1, dec2, dec3, and_I0, and_I1, and_I2, and_I3;
   decoder24 D0(dec0, dec1, dec2, dec3, s0, s1);
   and(and_I0, dec0, I0);
   and(and_I1, dec1, I1);
   and(and_I2, dec2, I2);
   and(and_I3, dec3, I3);
   or(out, and_I0, and_I1, and_I2, and_I3);
endmodule
```



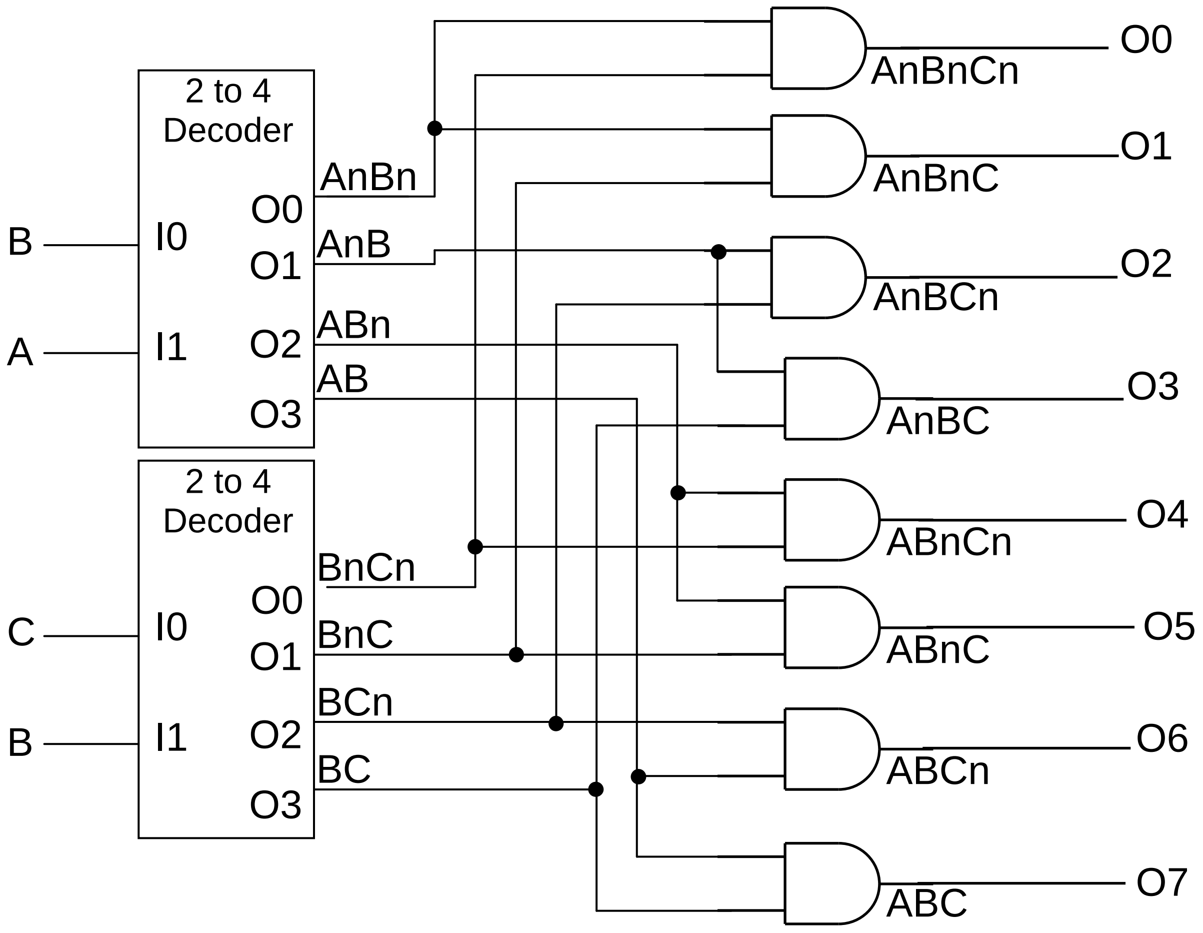
A.4. Implement a 3-input XOR gate using *not*, *and*, and *or* gates. Draw the schematic of the corresponding circuit.

```
module my_xor3(out, I0, I1, I2);
   input I0, I1, I2;
   output out;
   wire I0_not, I1_not, I2_not, int0, int2, int3, int4;
   not(I0_not, I0);
   not(I1_not, I1);
   not(I2_not, I2);
   and(int0, I0, I1_not, I2_not);
   and(int1, I0_not, I1, I2_not);
   and(int2, I0_not, I1_not, I2);
   and(int3, I0, I1, I2);
   or(out, int0, int1, int2, int3);
endmodule
```



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

```
module decoder38(O0, O1, O2, O3, O4, O5, O6, O7, A, B, C);
   input A, B, C;
   output O0, O1, O2, O3, O4, O5, O6, O7;
   wire AnBn, AnB, ABn, AB, BnCn, BnC, BCn, BC;
   decoder24 D0(AnBn, AnB, ABn, AB, B, A);
   decoder24 D1(BnCn, BnC, BCn, BC, C, B);
   and(O0, AnBn, BnCn);
   and(O1, AnBn, BnC);
   and(O2, AnB, BCn);
   and(O3, AnB, BC);
   and(O4, ABn, BnCn);
   and(O5, ABn, BnC);
   and(O6, AB, BCn);
   and(O7, AB, BC);
endmodule
```



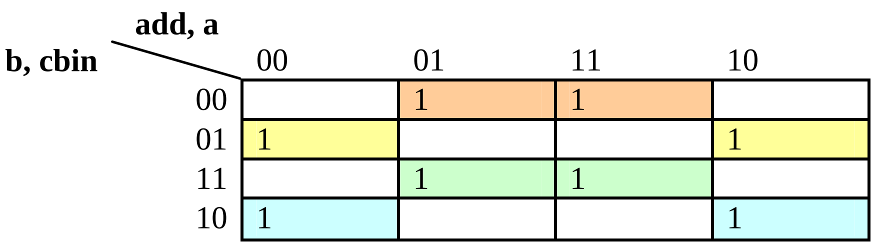
A.6. Implement a 1-bit adder/subtracter module using built-in gates. Use the method of Section 8.7 to construct it.

Let's represent the inputs to add/subtract as **a** and **b**. Call the carry/barrow input **cbin**. Call the control signal that indicates whether to add or subtract **add**, where **add** = 1 means we want to add and **add** = 0 means we want to subtract. The outputs will be the carry/barrow output **cbout** and the sum/difference **result**.

For an add function: **result** = **a** + **b** + **cbin**
For subtract function: **result** = **a** – **b** – **cbin**

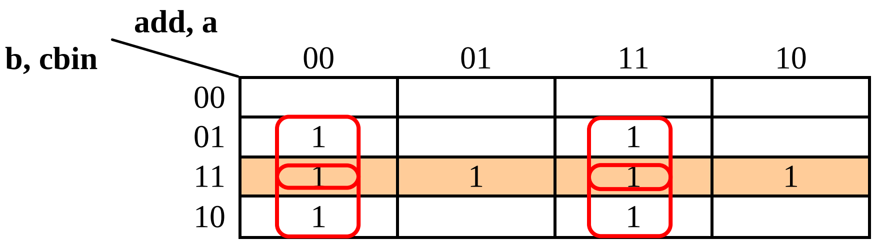
add	a	b	cbin	cbout	result
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	1
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	1	1

result



b, cbin \ add, a	00	01	11	10
00		1	1	
01	1			1
11		1	1	
10	1			1

cbout



b, cbin \ add, a	00	01	11	10
00				
01	1		1	
11	1	1	1	1
10	1		1	

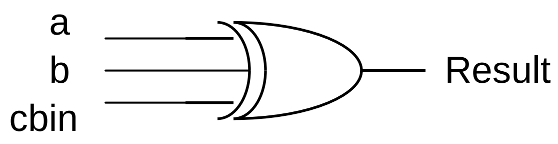
From the K-maps, we find the following minimized solutions:

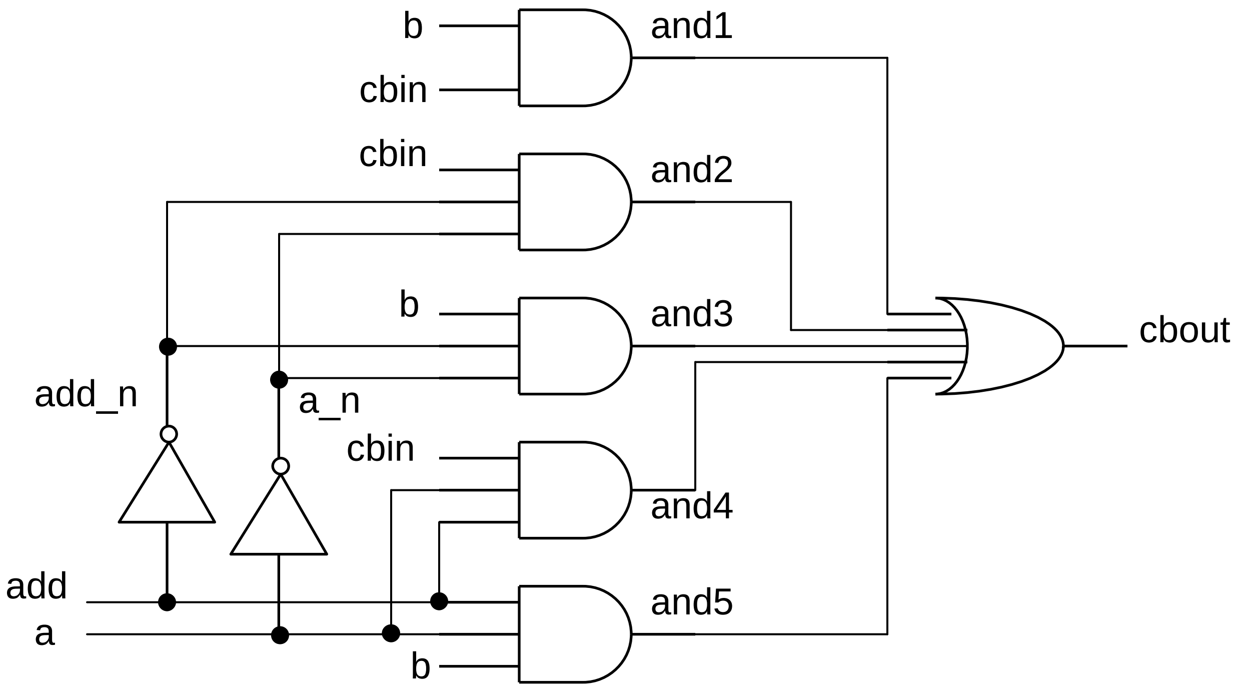
result = a'•b'•cbin + a'•b•cbin' + a•b'•cbin' + a•b•cbin = a ⊕ b ⊕ cbin

cbout = b•cbin + add'•a'•cbin + add'•a'•b + add•a•cbin + add•a•b

```
module add_sub(add, a, b, cbin, result, cbout);
   input add, a, b, cbin;
   output result, cbout;
   wire add_n, a_n, and1, and2, and3, and4, and5;

   xor(result,  a,  b, cbin);
   not(add_n, add);
   not(a_n, a);
   and(and1, b, cbin);
   and(and2, add_n, a_n, cbin);
   and(and3, add_n, a_n, b);
   and(and4, add, a, cbin);
   and(and5, add, a, b);
   or(cbout, and1, and2, and3, and4, and5);
endmodule
```

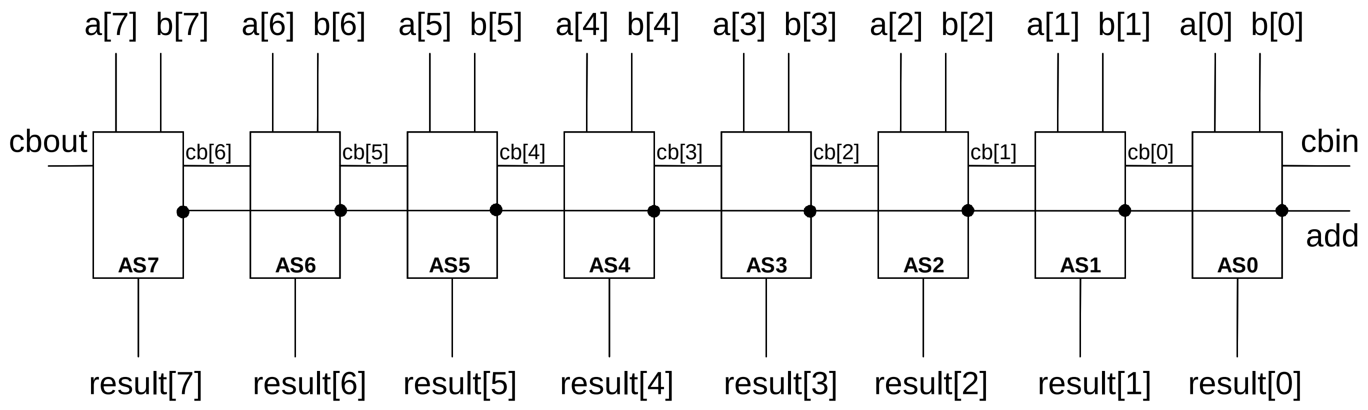




A.7. Implement an 8-bit adder/subtracter using the module you created in the previous problem.

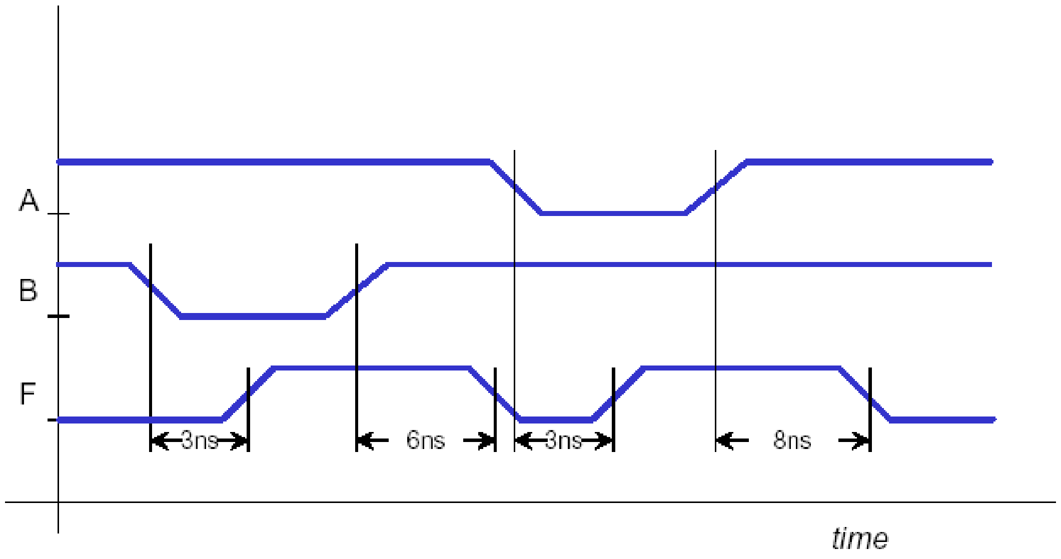
```
module add_sub8(a, b, cbin, add, result, cbout);
   input [7:0] a, b;
   input cbin, add;
   output [7:0] result;
   output cbout;
   wire [6:0] cb;

   add_sub AS0(add, a[0], b[0], cbin, result[0], cb[0]);
   add_sub AS1(add, a[1], b[1], cb[0], result[1], cb[1]);
   add_sub AS2(add, a[2], b[2], cb[1], result[2], cb[2]);
   add_sub AS3(add, a[3], b[3], cb[2], result[3], cb[3]);
   add_sub AS4(add, a[4], b[4], cb[3], result[4], cb[4]);
   add_sub AS5(add, a[5], b[5], cb[4], result[5], cb[5]);
   add_sub AS6(add, a[6], b[6], cb[5], result[6], cb[6]);
   add_sub AS7(add, a[7], b[7], cb[6], result[7], cbout);
endmodule
```


a[7] b[7] a[6] b[6] a[5] b[5] a[4] b[4] a[3] b[3] a[2] b[2] a[1] b[1] a[0] b[0]
cbout
cbin
add
cb[6] cb[5] cb[4] cb[3] cb[2] cb[1] cb[0]
AS7 AS6 AS5 AS4 AS3 AS2 AS1 AS0
result[7] result[6] result[5] result[4] result[3] result[2] result[1] result[0]

ECEn/CS 224
Chapter 10 Homework Solutions

10.1 The figure below is a detailed timing diagram for a NAND-like gate. If you were interested in knowing how fast this gate would run for purposes of critical path analysis, what would you use for t_{prop} ? Why?



The critical path of the gate is the worst case delay and should be used for t_{prop}. In this case, the worst case delay is 8 ns, thus t_{prop} = 8 ns.

10.2 If you were doing a more detailed timing analysis which separated out $t_{prop\text{-}rise}$ and $t_{prop\text{-}fall}$, what would be reasonable values to use for each of these delays?

We use the worst case delay for each of these. The worst case for $t_{prop\text{-}rise}$ is 3ns. The worst case for $t_{prop\text{-}fall}$ is 8ns.

10.3. Determine what the critical path is through circuit of Figure 10.15. To answer this question, redraw the circuit for your answer and draw a heavy line along the critical path. NOTE: inverters are often not drawn in schematics to simplify them. However, any signal that is complemented (*A'*) has an implied inverter on it which *must* be accounted for in any delay calculations, so draw them in when you redraw the circuit. Use the delays in Table 10.1 and be sure to give the total delay you calculated.

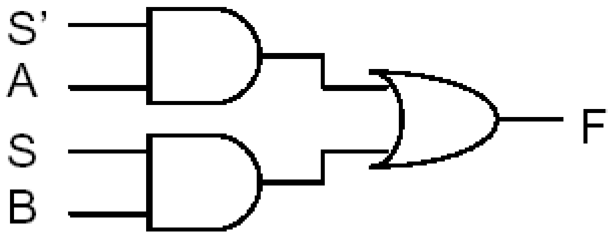
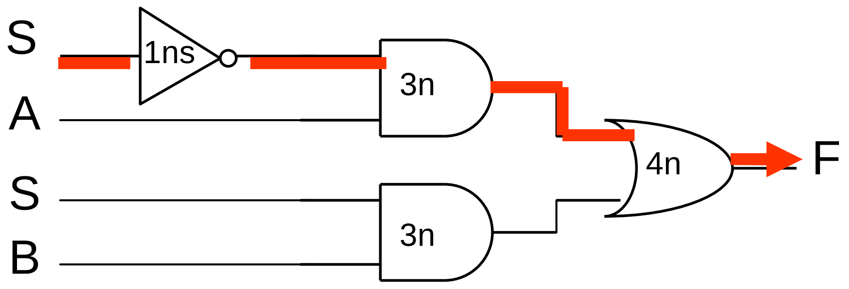


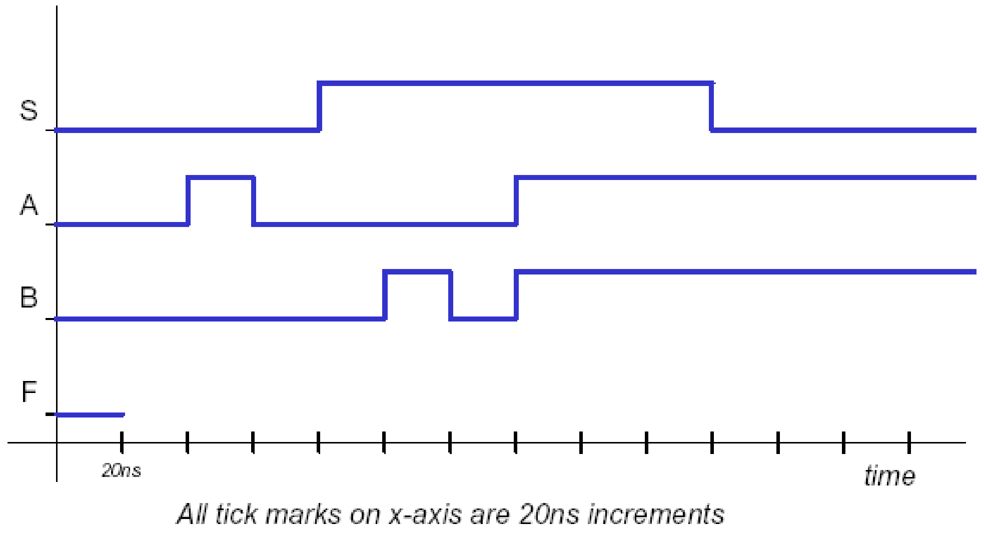
Figure 10.15

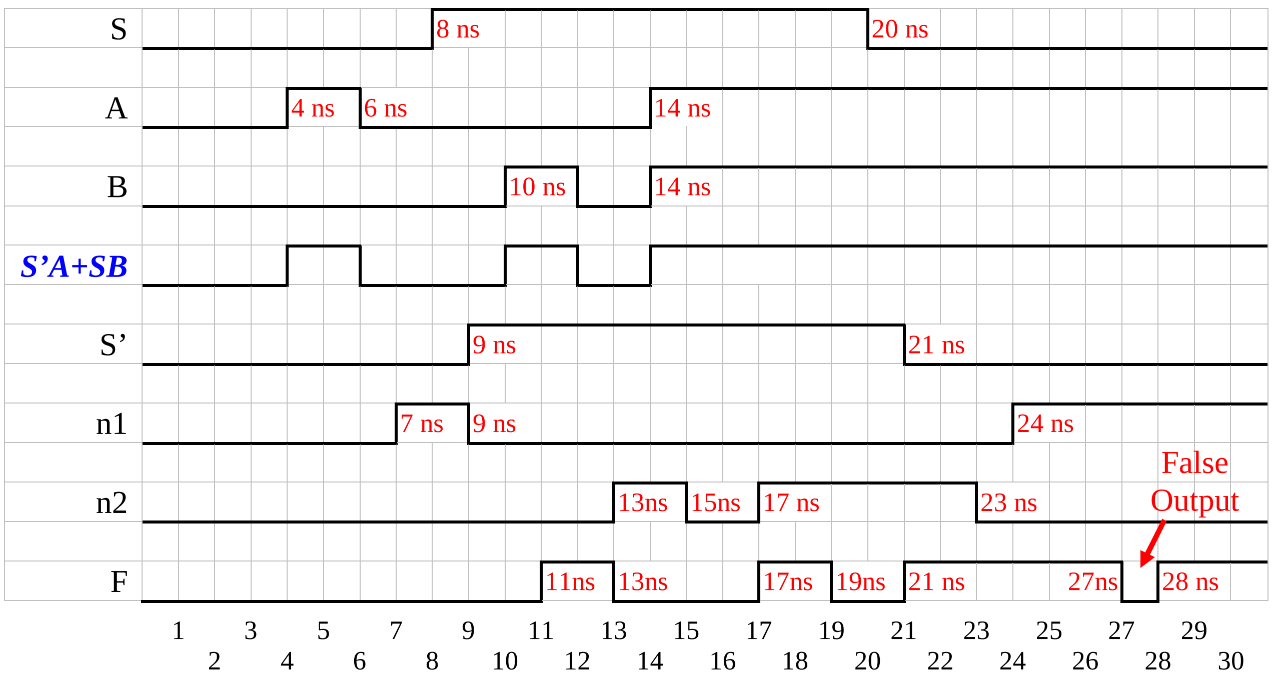
Gate Name	Delay
NOT	1ns
AND2	3ns
AND3	5ns
AND4	7ns
NAND2	2ns
NAND3	4ns
NAND4	6ns
OR2	4ns
OR3	6ns
OR4	8ns
NOR2	3ns
NOR3	5ns
NOR4	6ns

Table 10.1



10.5 For the circuit of Figure 10.15, complete the timing diagram of Figure 10.16. In your timing diagram, clearly label the time of each signal transition. Does the timing diagram contain any false outputs? Use the delays in the table 10.1.





The graph of ***S'A+SB*** has been drawn to show the ideal logic behavior without gate delays. We see that F is a delayed version of ***S'A+SB***, with the exception of a false output at 27 ns.

10.6 Determine what the critical path is through the circuit of Figure 10.17. To answer this question, redraw the circuit for your answer and draw a heavy line along the critical path. NOTE: inverters are often not drawn in schematics to simplify them. However, any signal that is complemented (*A'*) has an implied inverter on it which *must* be accounted for in any delay calculations. Use the delays from the table above and be sure to give the total delay you calculated.

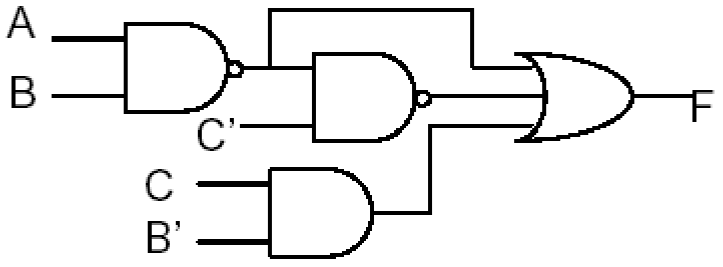
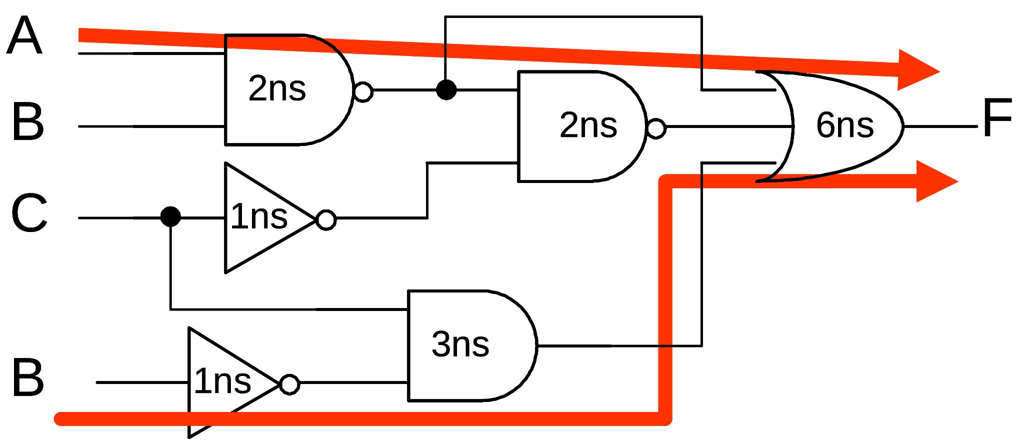


Figure 10.17



This circuit has multiple equal paths with the following delays:

$$t_{NAND2} + t_{NAND2} + t_{OR3} = 2\text{ns} + 2\text{ns} + 6\text{ns} = 10\text{ns}$$

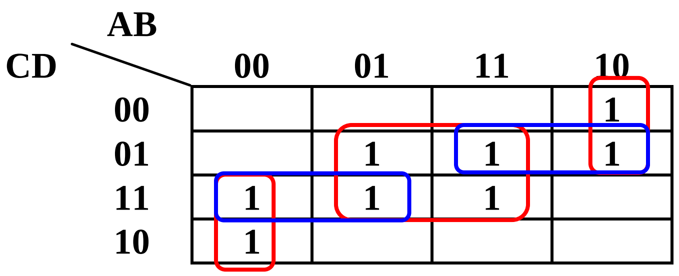
or

$$t_{NOT} + t_{AND2} + t_{OR3} = 1\text{ns} + 3\text{ns} + 6\text{ns} = 10\text{ns}$$

10.7 Consider the following function:

$$F(A,B,C,D) = \sum m(2, 3, 5, 7, 8, 9, 13, 15)$$

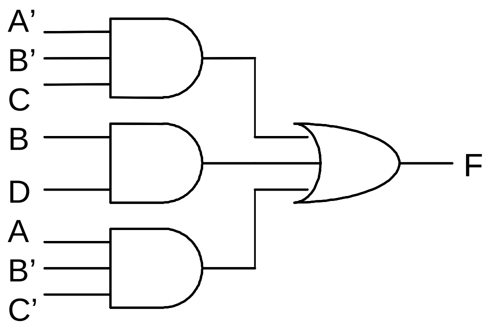
Draw a 4-variable KMap for this function. Then, do a conventional minimization of that KMap and implement the equation using gates. Finally, do a hazard-free minimization and implement those equations using gates. For each hazard, how many gates are added to the design?



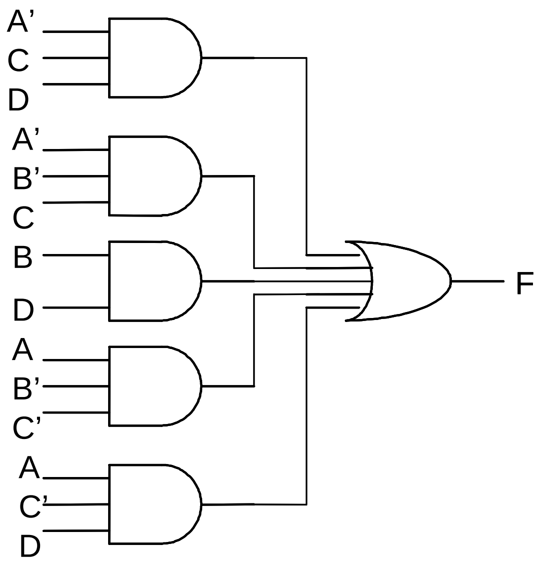
CD \ AB	00	01	11	10
00				1
01		1	1	1
11	1	1	1	
10	1			

Normal minimization (red) = A’B’C + BD + AB’C’
Hazard free design (red and blue) = A’B’C + BD + AB’C’ + A’CD + AC’D

Notice that for each hazard we added one prime implicant. This results in one added gate per hazard.



Minimized



Hazard Free