ECCS 1721 Digital Logic

Exam 2

Directions: PLEASE READ THESE FIRST!!!

This exam is *open-notebook* (meaning all of your course materials may be consulted), and the internet may be used to consult information on any of the topics; however, you may not use the internet or any other means of communication to discuss details of the exam with anyone else ***except the instructor***. Artificial intelligence application are not allowed. I repeat, **NO form of communication is allowed between students concerning any aspect of the exam**. ***Violating these directions is considered as giving or receiving aid on the exam***. **Please, ask the instructor for clarifications as needed**.

Honor Pledge

(Please copy down the *italicized statement* below on the first sheet you use in writing out your exam and sign before you begin the exam): **[10 pt penalty if omitted!!!]**

I hereby pledge my honor that no aid shall be given nor received during this exam. If I observe any forms of academic misconduct or questionable behavior, I hereby pledge to report it to my instructor.

Student Signature: [illegible]

You may either print this exam or just use separate sheets of paper; however, if using separate paper, please Indicate the beginning of each Problem of the exam (e.g., Problem 1) and be sure to number the answers appropriately (e.g., 3b). Use the same numbering as this exam.

Scan the completed exam and submit it to Canvas as a single pdf file.

Question	Score	Out of
1		12
2		13
3		15
4		12
5		16
6		14
7		18
Total		**100**

Question 1 (12 points)
Substitution blocks are used in block encryption techniques to substitute numbers with arbitrary values. The following truth table is for a 3-bit substitute block:

I2	I1	I0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

O2	O1	O0
0	1	1
0	1	0
0	0	0
1	1	1
1	1	0
0	0	1
1	0	0
1	0	1

1. **(3 points)** Construct three input K-maps for all three outputs.
2. **(3 points)** Find the product-of-sum expressions (Maxterms) for all outputs. **Hint:** you are grouping the zeros, and then you are applying Demorgan's Law to find the final output.
3. **(3 points)** Draw the block diagram of this substitute block using only 2-input NOR gates and inverters. **Hint:** you can use three inverters at the inputs to get the inputs as active high or low and utilize that for all outputs as we did in class.
4. **(2 points)** What is the total number of transistors needed to implement this hardware, assuming only 2-input NOR gates and not counting the three inverters at the inputs.
5. **(1 points)** Any substitution block at the transmitter side has to be inverted at the receiver side. Since this substitution block is one-to-one, it can be easily inverted by assuming the outputs as inputs and vice versa. Find the truth table of the inverse substitution block used at the receiver side.

Question 2 (13 points)
The following truth table illustrates a hardware module featuring four inputs labeled I3 to I0 and four outputs labeled O3 to O0. This hardware module is designed to convert signed numbers into unsigned numbers. When the most significant bit of the input is 0, the output mirrors the input,

whereas if the most significant bit of the input is 1, the 2’s complement of the input is transmitted to the output:

I3	I2	I1	I0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

O3	O2	O1	O0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
0	1	1	1
0	1	1	0
0	1	0	1
0	1	0	0
0	0	1	1
0	0	1	0
0	0	0	1

1. **(3 points)** Find the Boolean expression of O0 and O3 by inspection.
2. **(4 points)** Using a four-input k-map find the simplified sum-of-product (Minterms) expression of O1 and O2.
3. **(4 points)** Draw the hardware of this module using invertors and multiple input And/Or gates (if you need, you can go up till four inputs per gate).
4. **(2 points)** This hardware module is the most optimized in terms of speed, power efficiency, and area usage for a four-bit input. Nevertheless, it lacks modularity due to the complexity involved in expanding it to accommodate a greater number of input bits. We have learned two different methods for determining the 2’s complement of numbers. Selecting one of these two methods, explain in your own words how would you utilize it to find the absolute value of m-bit inputs.

Question 3 (15 points)

The following is the truth table for a 2-to-1 priority encoder, where O represents the single bit output and V indicates if the output is Valid

I1	I0
0	0
0	1
1	0
1	1

V	O
0	X
1	0
1	1
1	1

1. **(2 points)** Using inspection, what is the most simplified Boolean expressions for both V and O. **Hint:** you can assume the don’t care output to be anything you want as long as it simplifies your expression.
2. **(2 points)** Draw the hardware of this 2-to-1 priority encoder.
3. **(5 points)** Based on what we studied in class:
    a. What is the truth table for a 2-to-1 multiplixer?
    b. What is the Boolean expression for a 2-to-1 multiplexer?

   c. What is the hardware for a 2-to-1 multiplexer?
4. **(5 points)** Based on what we studied in class:
   a. What is the truth table for a 2-to-4 decoder? **Hint:** this is the simple decoder that transforms n to 2^n
   b. What are the Boolean expressions of the 2-to-4 decoder?
   c. What is the hardware for the 2-to-4 decoder?
5. **(1 points)** Rank those three basic components from most complicated to least complicated hardware. Explain why?

Question 4 (12 points)

The following table is driven from a waveform of inputs that change between Low '0' and high'1' at different 100 ns time steps from 0 till 800 ns. The first input is a data input of a 1-to-4 demultiplexer and the last two inputs are the select inputs of the demultiplexer.

1. **(8 points)** Fill in the remaining rows of the table for the different outputs. **Hint:** when an output is not selected it goes back to L.
2. **(4 points)** Based on your solution draw the waveforms of the different outputs.

	0-100 ns	100-200 ns	200-300 ns	300-400 ns	400-500 ns	500-600 ns	600-700 ns	700-800 ns
Input	H	L	L	H	H	L	L	H
Sel1	L	L	L	H	H	H	H	L
Sel0	L	L	H	H	L	H	L	H
Output3								
Output2								
Output1								
Output0								

Question 5 (16 points)

Given a 1024-bit Carry Look-ahead adder that groups every four bits together as described in class.

1. **(2 points)** How many partial full adders would this adder require?
2. **(3 points)** Show the block diagram and Boolean expressions of the partial full adder and explain in your own words what does it do?
3. **(2 points)** How many CLA blocks would this adder require?
4. **(5 points)** Show the Boolean expressions of the CLA block and explain in your own words what does it do? **Hint:** a general CLA will have one carry-in, four propagate inputs, and four generate inputs, and it generates three carry-outs, and one group propagate, and one group generate outputs. You can give me the Boolean expressions assuming: C_0, P_0, P_1, P_2, P_3, G_0, G_1, G_2, and G_3 are your inputs, and C_1, C_2, C_3, P_{0to3}, and G_{0to3} are your outputs.
5. **(2 points)** How fast in terms of t_g can this adder process the 1024 bits?
6. **(1 point)** What applications would require such an adder that clearly is very complicated and consumes a lot of power?
7. **(1 point)** Which one of the three adders we studied in class is considered a good compromise, that is consumes moderate amount of power, requires moderate number of gates, and achieves moderate speed? Why?

Question 6 (14 points)

1. **(9 points)** Draw the hardware of a signed 4by4 array multiplier**. Hint:** you will be using inverters, AND gates, HA, FA, and cheap FA (FA*). Note that the last Full Adder of all the

ripple-carry-adders used in this multiplier is cheap because it only needs to generate one output.

2. **(3 points)** What is the total number of inverters, AND gates, HAs, FAs, and FA*s used?
3. **(2 points)** What is the estimated delay in t_g through this array multiplier, please show your critical path using a different color on your hardware diagram in 1.

Question 7 (18 points)

An alternative of comparing two unsigned numbers X and Y is to:

- find D=X-Y
- Check the sign of D to determine if X<Y or X≥Y.
- Use a multi-input Nor gate to detect if D=0 and hence X=Y.

To subtract Y from X and find D, we can use a ripple-borrow-subtractor. Similar to the ripple-carry-adder, the ripple-borrow subtractor ripples the borrow from the least significant bit slice to the most significant bit slice. That is the Borrow out (B_{out}) of a bit slice is sent to the borrow in (B_{in}) of the next bit slice. The least significant bit slice is implemented using a Half Subtractor (HS) and the remaining bit slices use Full Subtractors (FS). The result is in two's complement, where the last B_{out} is the sign bit.

The truth table of a HS is as follows:

X	Y
0	0
0	1
1	0
1	1

Bout	D
0	0
1	1
0	1
0	0

1. **(4 points)** Find the Boolean expressions of the two outputs B_{out} and D in terms of X and Y and draw the hardware. **Hint:** you can utilize Xor gates.

The truth table of a FS is as follows:

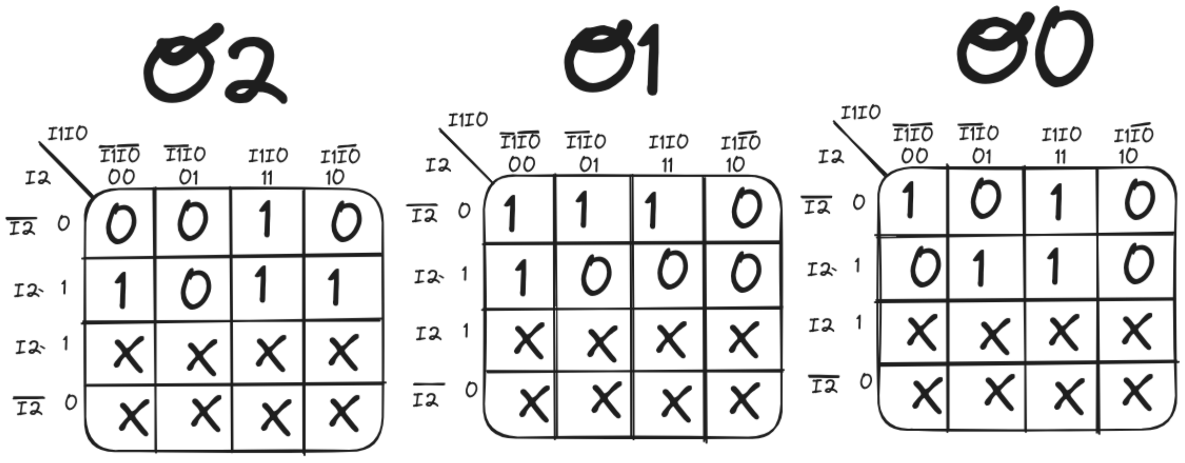
X	Y	B_{in}
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

B_{out}	D
0	0
1	1
1	1
1	0
0	1
0	0
0	0
1	1

2. **(5 points)** Find the Boolean expressions of the two outputs B_{out} and D in terms of X , Y, and B_{in} and draw the hardware. **Hint:** you can utilize Xor gates.
3. **(1 point)** The most significant bit slice of this ripple-borrow-subtractor will only have one output D since it does not need to generate a B_{out}. Draw the hardware of this FS^*.
4. **(5 points)** Draw two copies of the hardware of a four-bit ripple-borrow-subtractor using HS, FS and FS^*. On the first copy run the example of 13-10 and on the second copy run the example of 10-13 showing all the bits for X, Y, B_{out} and D for all bit slices.
5. **(3 points)** Draw the hardware of the whole four-bit unsigned comparator that will take X and Y as inputs, use the four-bit ripple-borrow subtractor and four-bit Nor gate and generate two single bit outputs (X<Y and X=Y).

Question 1

1.1:



O2

I2 \ I1I0	$\overline{I1}\overline{I0}$ 00	$\overline{I1}I0$ 01	$I1I0$ 11	$I1\overline{I0}$ 10
$\overline{I2}$ 0	0	0	1	0
I2 1	1	0	1	1
I2 1	X	X	X	X
$\overline{I2}$ 0	X	X	X	X

O1

I2 \ I1I0	$\overline{I1}\overline{I0}$ 00	$\overline{I1}I0$ 01	$I1I0$ 11	$I1\overline{I0}$ 10
$\overline{I2}$ 0	1	1	1	0
I2 1	1	0	0	0
I2 1	X	X	X	X
$\overline{I2}$ 0	X	X	X	X

O0

I2 \ I1I0	$\overline{I1}\overline{I0}$ 00	$\overline{I1}I0$ 01	$I1I0$ 11	$I1\overline{I0}$ 10
$\overline{I2}$ 0	1	0	1	0
I2 1	0	1	1	0
I2 1	X	X	X	X
$\overline{I2}$ 0	X	X	X	X

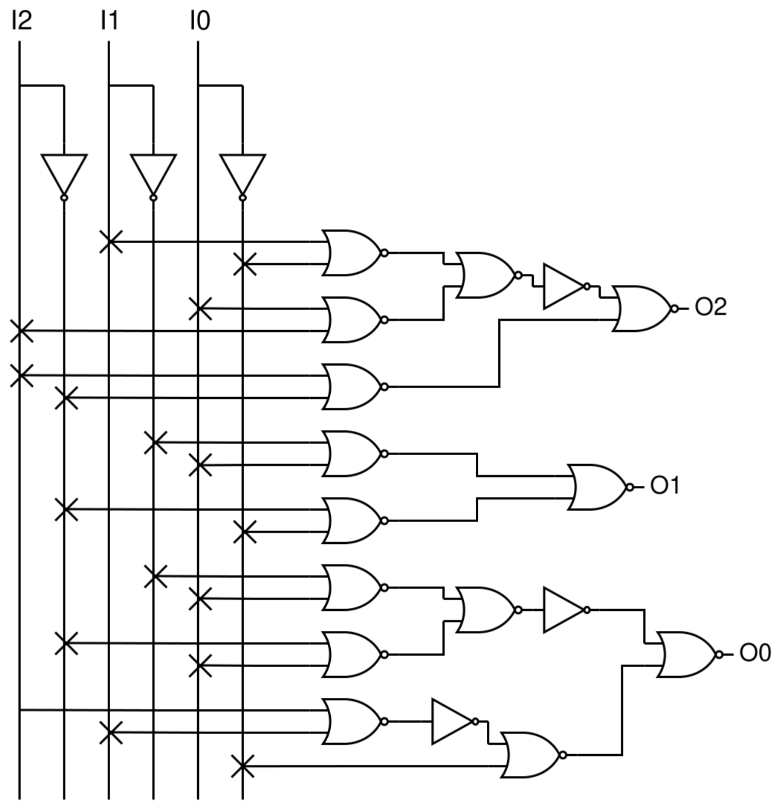
1.2:

$$\overline{O2} = \overline{I1}I0 + \overline{I2}\overline{I0} + \overline{I2}\overline{I1}$$
$$O2 = (I1 + \overline{I0})(I2 + I0)(I2 + I1)$$

$$\overline{O1} = I1\overline{I0} + I2I0$$
$$O1 = (\overline{I1} + I0)(\overline{I2} + \overline{I0})$$

$$\overline{O0} = I1\overline{I0} + I2\overline{I0} + \overline{I2}\overline{I1}I0$$
$$O0 = (\overline{I1} + I0)(\overline{I2} + I0)(I2 + I1 + \overline{I0})$$

1.3:



1.4: $14 \cdot 4\text{Ts} + 3 \cdot 2\text{Ts} = 62\text{Ts}$

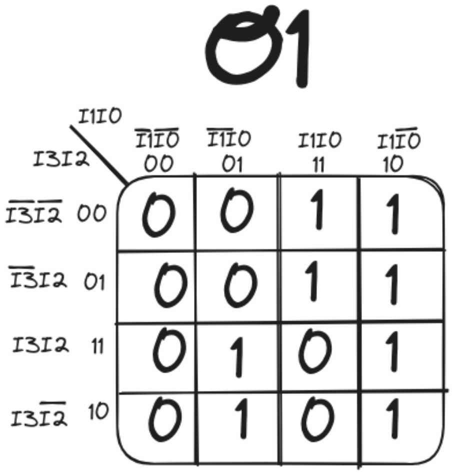
1.5:

I2	I1	I0	O2	O1	O0
0	1	1	0	0	0
0	1	0	0	0	1
0	0	0	0	1	0
1	1	1	0	1	1
1	1	0	1	0	0
0	0	1	1	0	1
1	0	0	1	1	0
1	0	1	1	1	1

Question 2

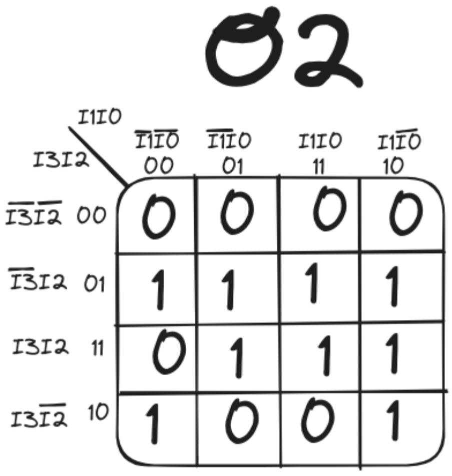
2.1: $O0 = I0 \qquad O3 = I3 \cdot \overline{I2} \cdot \overline{I1} \cdot \overline{I0}$

2.2:



O1

$I3I2 \backslash I1I0$	$\overline{I1}\,\overline{I0}$ 00	$\overline{I1}I0$ 01	$I1I0$ 11	$I1\overline{I0}$ 10
$\overline{I3}\,\overline{I2}$ 00	0	0	1	1
$\overline{I3}I2$ 01	0	0	1	1
$I3I2$ 11	0	1	0	1
$I3\overline{I2}$ 10	0	1	0	1

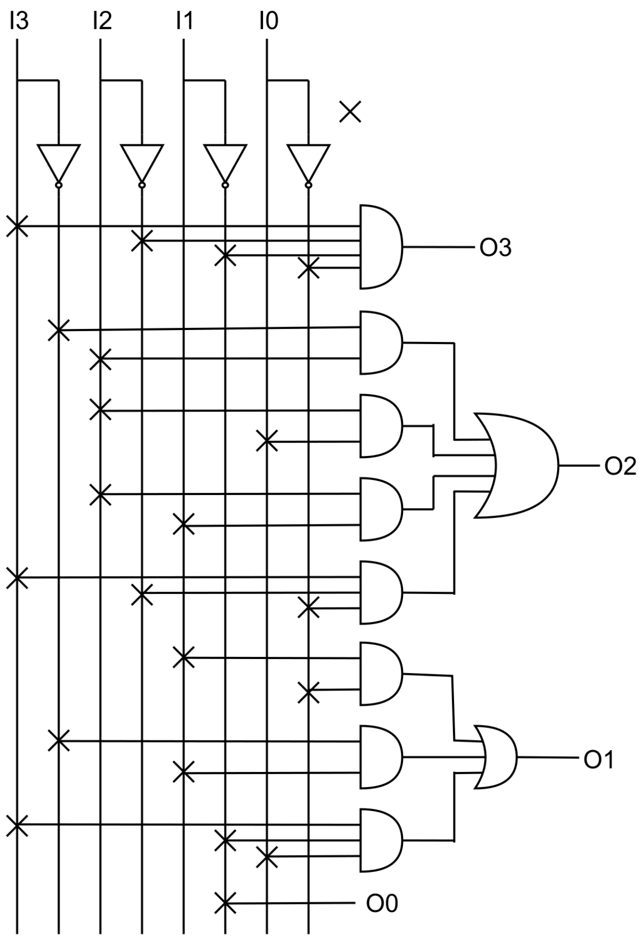


O2

$I3I2 \backslash I1I0$	$\overline{I1}\,\overline{I0}$ 00	$\overline{I1}I0$ 01	$I1I0$ 11	$I1\overline{I0}$ 10
$\overline{I3}\,\overline{I2}$ 00	0	0	0	0
$\overline{I3}I2$ 01	1	1	1	1
$I3I2$ 11	0	1	1	1
$I3\overline{I2}$ 10	1	0	0	1

$O1 = I1\overline{I0} + \overline{I3}I1 + I3\overline{I1}I0 \qquad O2 = \overline{I3}I2 + I2I0 + I2I1 + I3\overline{I2}\,\overline{I0}$

2.3:

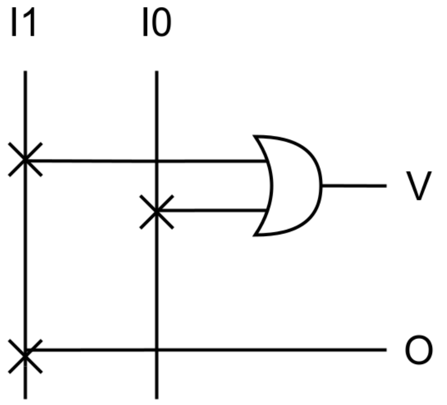


2.4: The method I will use for determing the absolute value of an m-bit input is by taking the 1's complement by inverting all bits through an m-bit inverter, then I will use a m-bit Carry Look-Ahead adder to add 1 to the number, thus giving us our absolute value. Though this idea is very expensive in transistors, it will be indeed modular and the Carry Look-Ahead adder will keep our Tg time low.

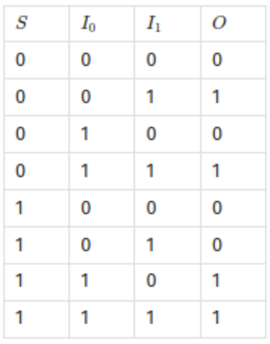
Question 3

3.1: $V = I1 + I0$ $O = I1$

3.2:



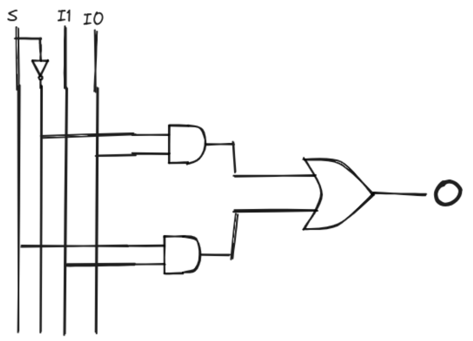
3.3a:



S	I_0	I_1	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

3.3b: $O = \overline{S}I_0 + SI_1$

3.3c:



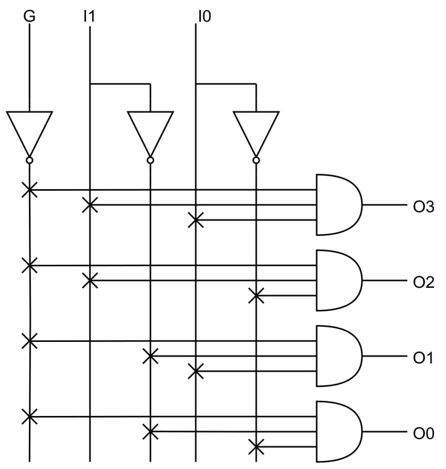
3.4a:

G	I1	I0	O3	O2	O1	O0
1	X	X	0	0	0	0
0	0	0	0	0	0	1
0	0	1	0	0	1	0
0	1	0	0	1	0	0
0	1	1	1	0	0	0

3.4b:

$$O3 = I1 \cdot I0 \cdot \overline{G}$$
$$O2 = I1 \cdot \overline{I0} \cdot \overline{G}$$
$$O1 = \overline{I1} \cdot I0 \cdot \overline{G}$$
$$O0 = \overline{I1} \cdot \overline{I0} \cdot \overline{G}$$

3.4c:



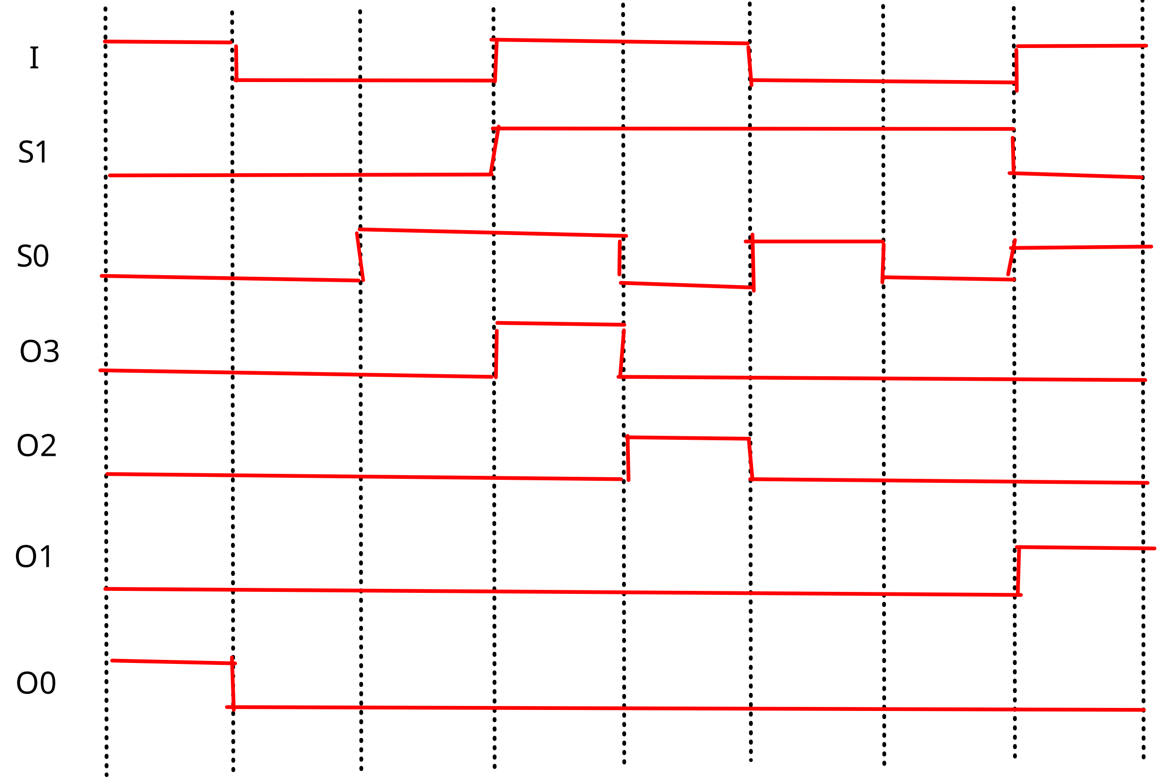
3.5: The most complicated hardware out of these three is the 2-4 decoder. It contains 4 3-input AND gates, which would actually be 8 AND gates in total, which is significantly moret than the other two. The next "medium" complicated hardware is the 2-to-1 multiplexer. It contains 2 AND gates and an OR gate, which is less than the decoder but more than the priority encoder. The least complicated hardware is the 2-to-1 priority encoder, which contains an entire 1 OR gate.

Question 4

4.1:

	0-100 ns	100-200 ns	200-300 ns	300-400 ns	400-500 ns	500-600 ns	600-700 ns	700-800 ns
Input	H	L	L	H	H	L	L	H
Sel1	L	L	L	H	H	H	H	L
Sel0	L	L	H	H	L	H	L	H
Output3	L	L	L	H	L	L	L	L
Output2	L	L	L	L	H	L	L	L
Output1	L	L	L	L	L	L	L	H
Output0	H	L	L	L	L	L	L	L

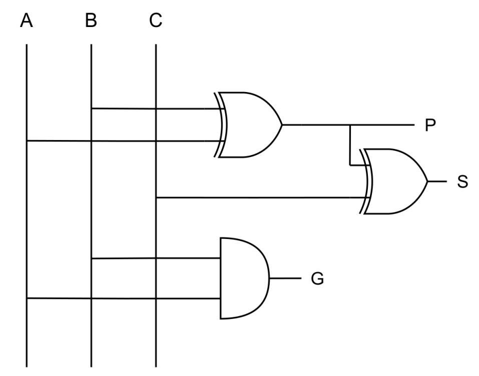
4.2:



Question 5

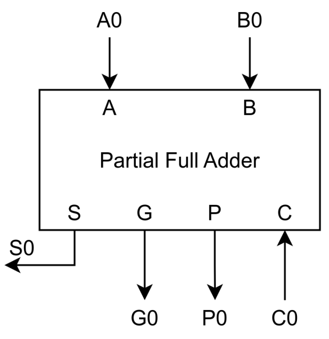
5.1: A 1024-bit carry look-ahead adder would require 1024 partial full adders.

5.2:



The partial full adder is a component of the carry look-ahead adder that can help us save time and reduce the ripple effect of carry. First, we need to find if A and B will themselves generate a carry, which is done with an AND gate, and we output as G. Then, we find if they can propagate a carry, which is done using either an OR or XOR gate, and we pass this as P. Then we can XOR P with our carry in, C, or we can XOR A and B again and XOR that with C, to get our sum S.

$$G_0 = A_0 \cdot B_0 \qquad P_0 = A_0 \oplus B_0$$
$$S_0 = (A_0 \oplus B_0) \oplus C_0 \quad \text{OR} \quad P_0 \oplus C_0$$



5.3: A 1024-bit carry look-ahead adder would require 256 4-bit CLA blocks.

5.4:

$$G_{0to3} = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$
$$P_{0to3} = P_3P_2P_1P_0$$
$$C_1 = G_0 + P_0C_0$$
$$C_2 = G_1 + P_1G_0 + P_1P_0C_0$$
$$C_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$$

The carry-look ahead logic will take the propagate and generate bits of each partial full adder and then use them to generate the carry bits that are passed to each subsequent partial full adder. Since every partial full adder has had the chance to produce their propagate and generate bits, the process is extremely fast, much faster than the ripple carry adder, which must wait for each carry bit to be produced.

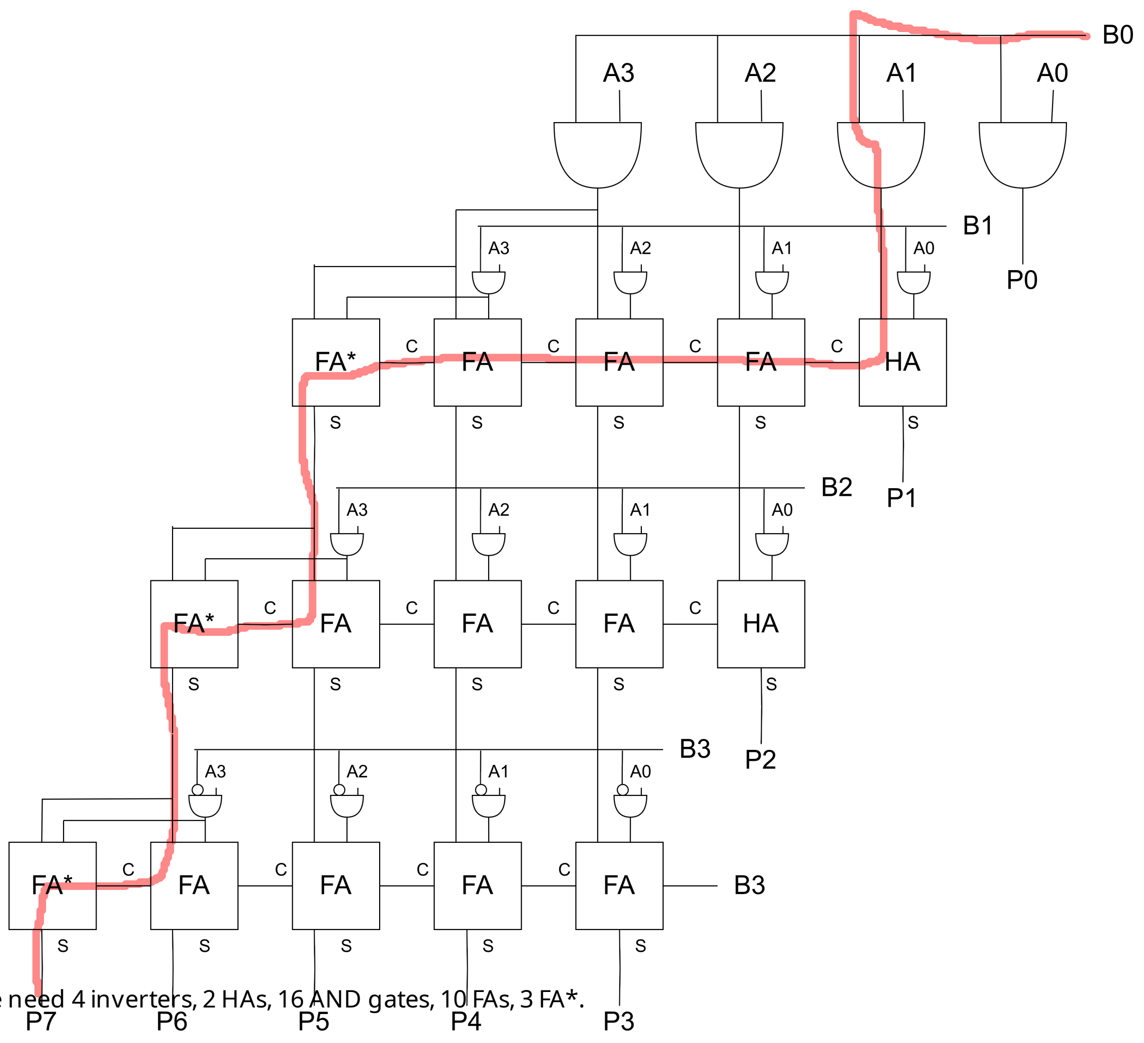
5.5: $tc : (log_2 1024)t_g + 2t_g = 10t_g + 2t_g = 12t_g$

5.6: In digital signal processing applications, a lot of data needs to be processed extremely quickly. Millions of bits need to be processed and potentially added together. Because of the logarithmic speed increase of carry look-ahead adders, this makes it perfect for these kind of applications.

5.7: The carry-select adder is a good compromise between the ripple carry adder and carry look-ahead adder. It works in parallel, and therefore makes it's critical pass much faster than that of the ripple carry adder, but unfortunately not as fast as the carry look-ahead adder. However, because of the carry-select logic required, it requires significantly more gates than a ripple carry adder, making it more costly, but not as costly as the carry look-ahead adder.

Question 6

6.1:



6.2: We need 4 inverters, 2 HAs, 16 AND gates, 10 FAs, 3 FA*.

6.3:

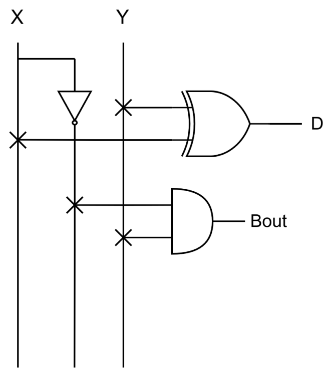
$$8 \cdot 2t_g + 1t_g + 1t_g = 18t_g$$

7.1:

Question 7

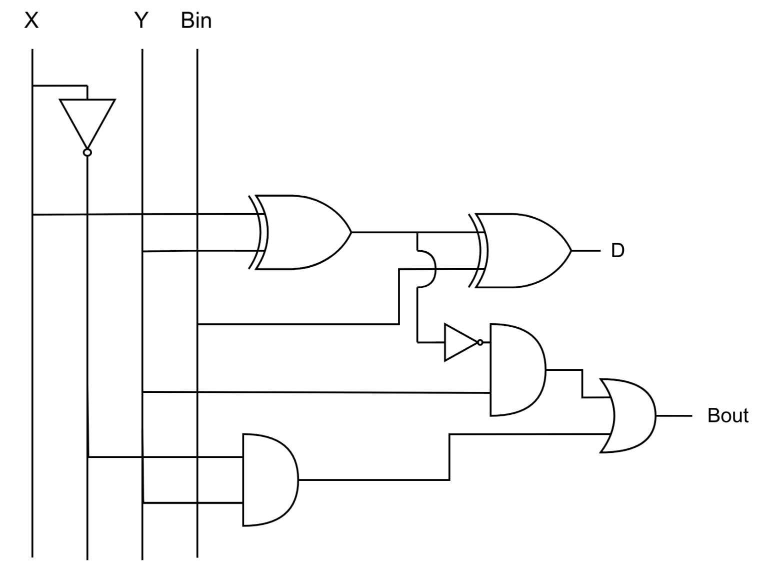
$$B_{out} = \overline{X} \cdot Y$$

$$D = X \oplus Y$$

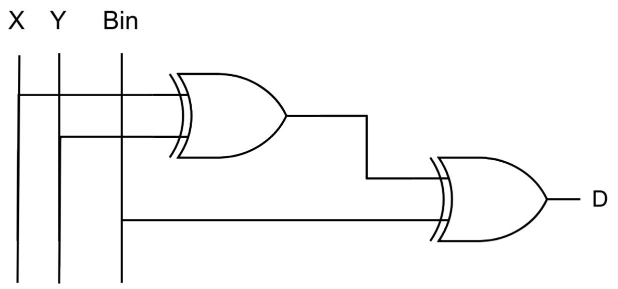


7.2: $B_{out} = B_{in} \cdot \overline{X \oplus Y} + \overline{A} \cdot B$

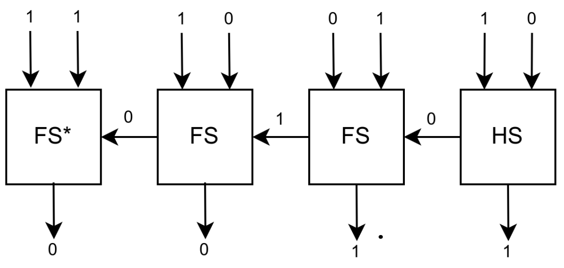
$D = (X \oplus Y) \oplus B_{in}$

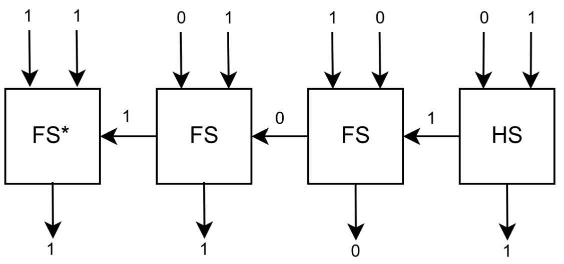


7.3:

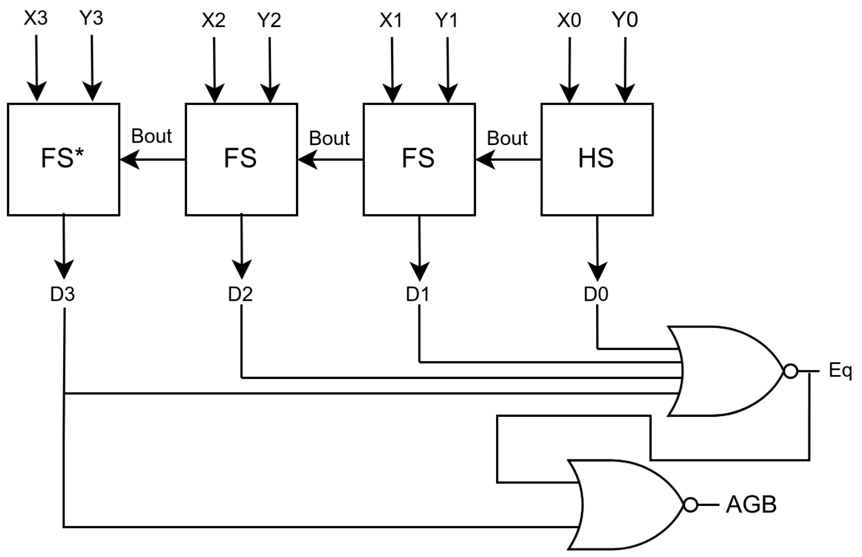


7.4:





7.5:


X3
Y3
X2
Y2
X1
Y1
X0
Y0
FS*
Bout
FS
Bout
FS
Bout
HS
D3
D2
D1
D0
Eq
AGB