Encoders, Decoders, Priority Encoders, Priority Decoders

1. Encoders
    a. Derive a minimized SOP expression using K-maps or Boolean Arithmetic theorems for a 2-to-4 decoder with enable before implementing it with our primitive gates.

    b. Implement a 3-to-8 decoder using using 2-to-4 decoders with enable.

c. Implement a 3-to-8 decoder using 2-to-4 decoders without enables.

d. Derive a minimized SOP expression using K-maps or Boolean Arithmetic theorems for a 4-to-2 encoder with enable, before implementing it with our primitive gates.

e. Implement an 8-to-3 encoder using 4-to-2 encoders.

2. Priority Encoders

a. Complete the following truth table for a priority encoder.

A_3	A_2	A_1	A_0	Y_1	Y_0	V
0	0	0	0			
0	0	0	1			
0	0	1	X			
0	1	X	X			
1	X	X	X			

b. Implement the following priority encoder using our primitive gates.

A_3	A_2	A_1	A_0	Y_1	Y_0	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

c. The following truth table describes the behavior of a special encoder. Based on the don't cares, list the inputs from highest to lowers priority (1-4).

A_3	A_2	A_1	A_0	Y_1	Y_0	V
0	0	0	0	X	X	0
X	0	X	1	0	0	1
0	0	1	0	0	1	1
X	1	X	X	1	0	1
1	0	X	0	1	1	1

(1)

(2)

(3)

(4)

d. Implement the combinational logic for output Y_1 for this special encoder using primitive gates.

Multiplexers and Demultiplexers

1. Multiplexers:
    a. Fill out a truth table for a 2-to-1 multiplexer. Write the SOP and POS expressions for the truth table.

    b. Design a 2-to-1 multiplexer using primitive gates.

    c. Implement a 2-to-1 multiplexer in Verilog using the provided skeleton module code below.

```
module Mux2_1b
(
    input  wire in0,
    input  wire in1,
    input  wire sel,
    output wire out
);

// Implement a 1-bit 2-to-1 multiplexer using gate level
modeling or Boolean equations




endmodule
```

d. Fill out a truth table for a 4-to-1 multiplexer and implement it using the minimal amount of primitive gates.

e. Implement the 4-to-1 multiplexer using primitive gates in Verilog using the provided skeleton module code below.

```
module Mux4_1b
(
    input  wire [3:0] in,
    input  wire [2:0] sel,
    output wire out
);

// Implement a 1-bit 4-to-1 multiplexer using gate level
modeling or Boolean equations











endmodule
```

f. Implement a 4-to-1 multiplexer in Verilog using the provided skeleton module code below. This time reuse the Mux2_1 module from Part C.

```
module Mux4_1b
(
    input  wire [3:0] in,
    input  wire [2:0] sel,
    output wire out
);

// Implement a 1-bit 4-to-1 multiplexer using Mux2_1b.










endmodule
```

2. Demultiplexer
    a. Fill out a truth table for a 1-to-2 demultiplexer. Write the SOP and POS expressions for the truth table.

    b. Design a 1-to-2 demultiplexer using primitive gates.

    c. Implement a 1-to-2 demultiplexer in Verilog using the provided skeleton module code below.

```
module Demux2_1b
(
  input wire in0,
  input wire sel,
  output wire [1:0] out
);

// Implement a 1-bit 1-to-2 demultiplexer using gate level
modeling or Boolean equations




endmodule
```

d. Design a 1-to-4 demultiplexer using the minimal amount of primitive gates.

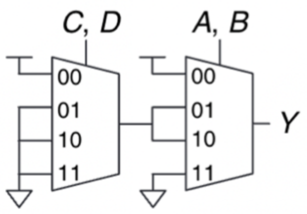
e. Implement a 1-to-4 multiplexer in Verilog using the provided skeleton module code below. Use the demux_1to4 module from Part C.

```
module Demux4_1b(
  input wire in,
  input wire [1:0] sel,
  output wire [3:0] out,
)

// Implement a 1-bit 1-to-4 demultiplexer using gate level
modeling or Boolean equations









endmodule
```

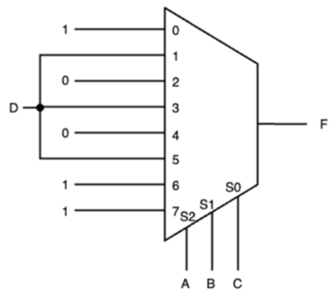
3. Below we will multiplexers to implement logic functions.
    a. Implement a 2-input OR gate using only 2-to-1 multiplexers.

    b. Implement the following Boolean expression using only 2-to-1 multiplexer(s):
    $F = AB + C + D$

c. Write a minimized Boolean equation for the function performed by the circuit in the following figure.



d. Derive and simplify the Boolean expression for the circuit based on an 8-to-1 multiplexer shown below.



Adders and Subtractors

1. Implement a one-bit full adder with carry in and carry out using primitive gates.
   a. Implement a one-bit full adder with carry in and carry out using primitive gates

2. Next, we will design three adders using 8-bit blocks and 2-input gates. For each adder, first draw the implementation of the 8-bit block. It may use one-bit full adders and 2-input primitive logic gates. Then compose multiple 8-bit blocks with any additional combinational logic to implement the adder. After designing the adder, label the critical path with its associated propagation delay as well as the short path with its associated contamination delay. Assume that our primitive gates have the following propagation and contamination delays:

Gate	t_{pd}	t_{cd}
NOT	1τ	1τ
AND2	5τ	4τ
OR2	3τ	2τ

a. 64-bit ripple carry

b. 64-bit carry-select

c. 64-bit carry-lookahead

3. Why might a designer choose to use a ripple-carry adder instead of a carry-lookahead?

4. Implement a 32-bit subtractor that computes the difference $Y = A - B$. You may only use 8 bit blocks and 2 input gates. Assume that $A \geq B$.

5. Implement a comparator for