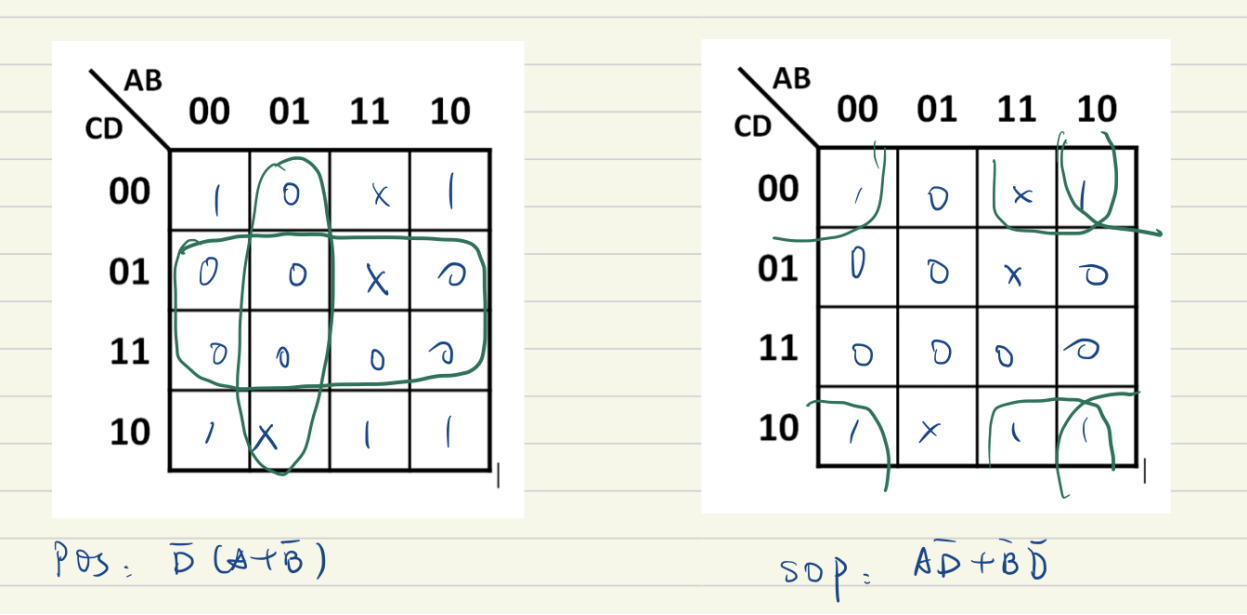
Q1. What is the decimal value of unsigned binary number 1100011? [3 pts]

1 + 2 + 32 + 64 = 99

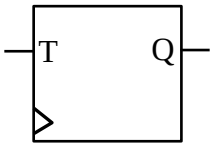
Q2. Use Karnaugh maps (K-maps) to simplify the **sum of products** and **product of sums** for the function $F = \sum_{ABCD}(0, 2, 8, 10, 14) + dc(6, 12, 13)$. Please fill out the K-maps, include the circles, and write down the final result for each K-map. [8 pts]

(a) Sum of Products
(b) Product of Sums



NetID:_____

Q3. In our class we have learned how to use a D flip-flop (DFF) to build a toggle flip-flop (TFF). Do you think we can use a TFF to implement a DFF? If so, please add to the following circuit to create a functional DFF. You need to clearly label the input(s) and output. You can introduce additional gates if needed. Otherwise, please concisely explain why this is not feasible. [5 pts]



DFF from TFF; T = f(D, Q)

Q_t	Q_{t+1}	D	T
0	0	0	0
0	1	1	1
1	0	0	1
1	1	1	0

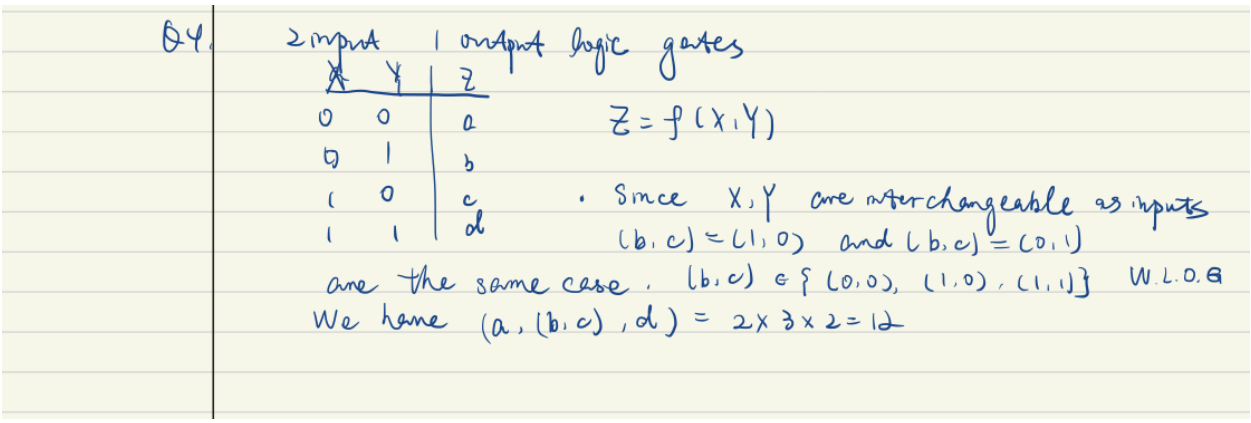
We need to design the circuit to generate triggering signal T as a function of D, Q

$T = D Q' + D' Q$ ⇒ Add an XOR gate

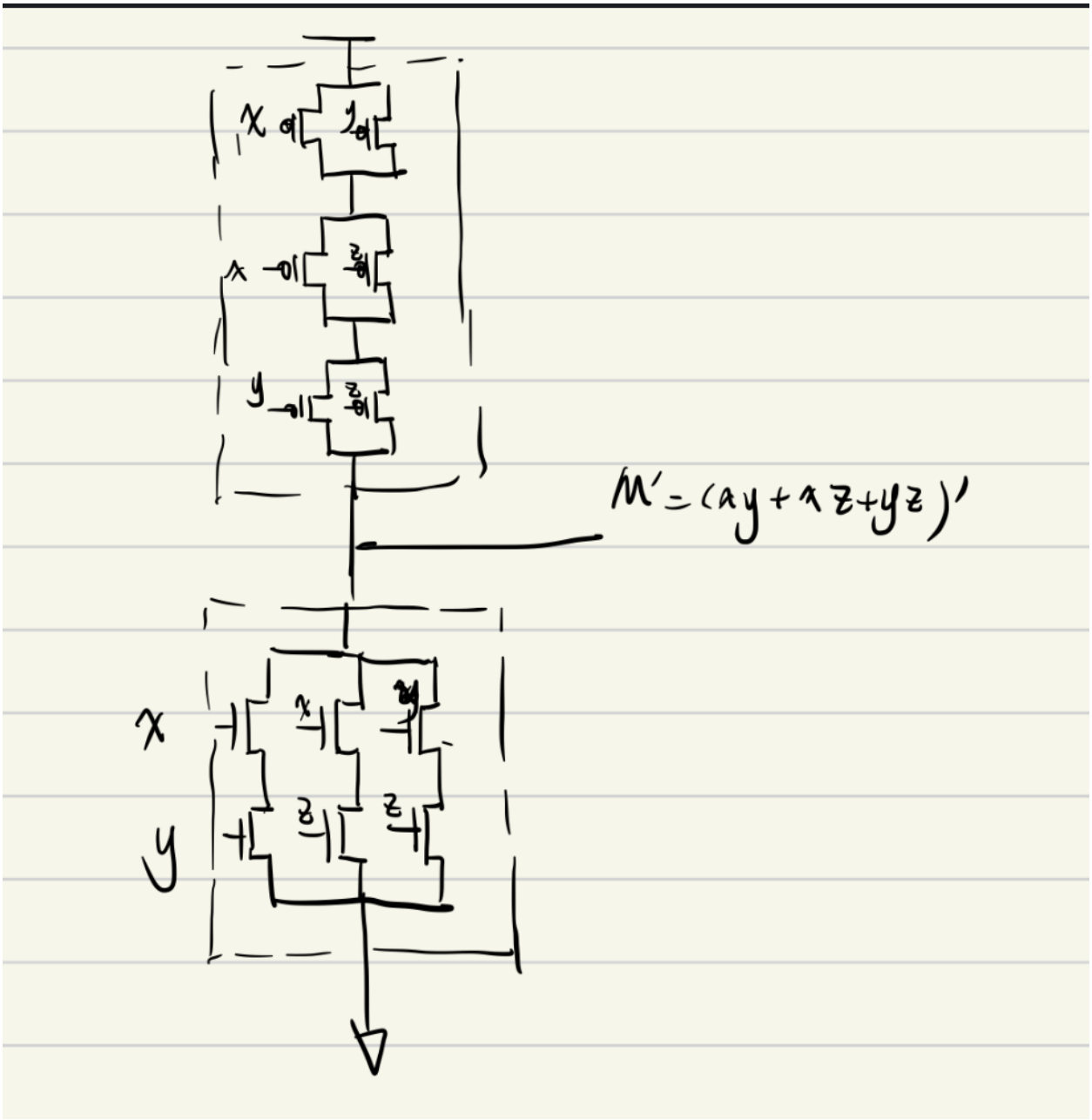
Q4. So far we have seen several different two-input logic gates, such as AND, OR, NAND, NOR, XOR, and XNOR. How many **distinct two-input one-output logic gates** in total do you think exist? Note that some of them may not have a name. Please concisely explain your reasoning and try to avoid exhaustive enumeration. [5 pts]

We have 4 output entries in the truth table of a 2-input gate. Each entry has two choices (0/1). Hence total is 2^4 = 16.

An alternative answer when we further consider input permutation:



Q5. A majority gate is a three-input, one-output logic gate, denoted as M(x, y, z). The output of a majority gate is high if and only if the majority (two or more) of the inputs are asserted high, that is, M(x, y, z) = xy + xz + yz. Please complete the CMOS circuit diagram for the majority gate. [9 pts]



Q6. Figure 6(a) shows the circuit diagram of a 3-input Boolean function, which is named as "C-gate" for the sake of convenience:

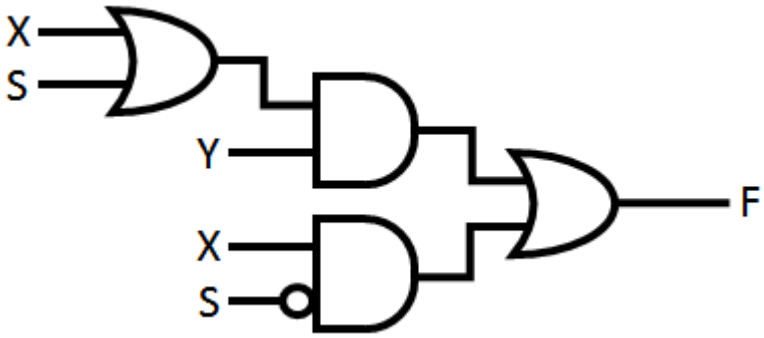


Figure 6(a) C-gate

(a) Please write down the Boolean expression of the C-gate. Do not make any simplifications to the expression. [2 pts]

F = (X + S)Y + XS'

(b) Please use Boolean algebra theorems to show that a C-gate is equivalent to a 2-to-1 multiplexer. [3 pts]

F = XY + SY + XS' = SY + S'X (consensus)

(c) Can you implement a D latch using **a minimum number of C-gates**? No other logic gates are allowed. The following symbols in Figures 6(b) and (c) represent the D latch and C-gate, respectively. [5 pts]

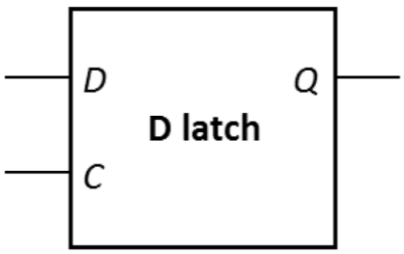


Figure 6(b)

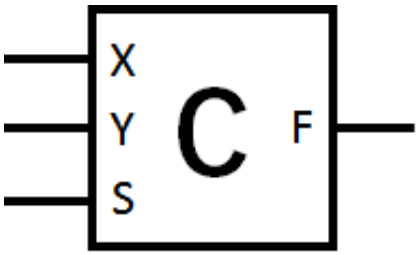
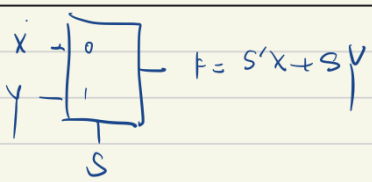


Figure 6(c)

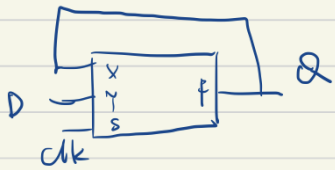
F is connected to X and D is connected to Y, CLK is connected to S

NetID:_____

Q6



Latch: $Q = Clock \cdot D + \overline{Clock} \cdot Q$



Q7. Consider the following circuit shown in Figure 7(a), which we name as "FOO". Here CLK1 is connected to the system clock, which toggles its level every cycle.

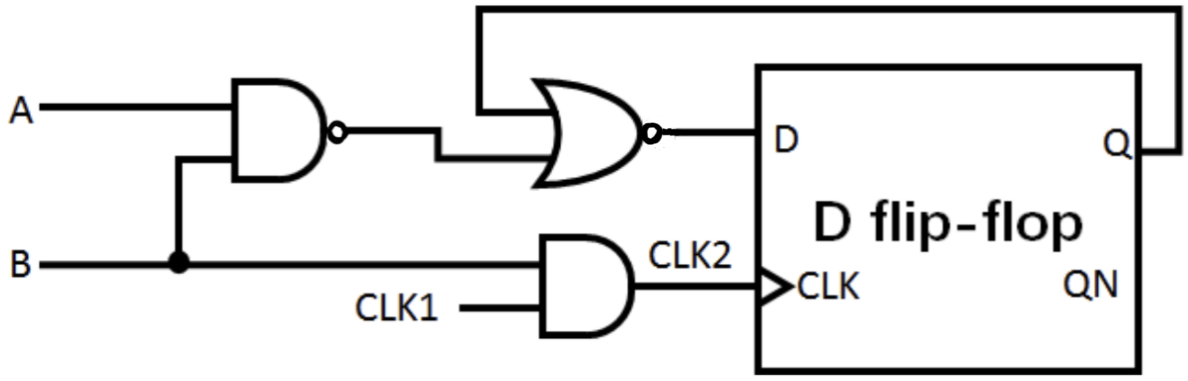
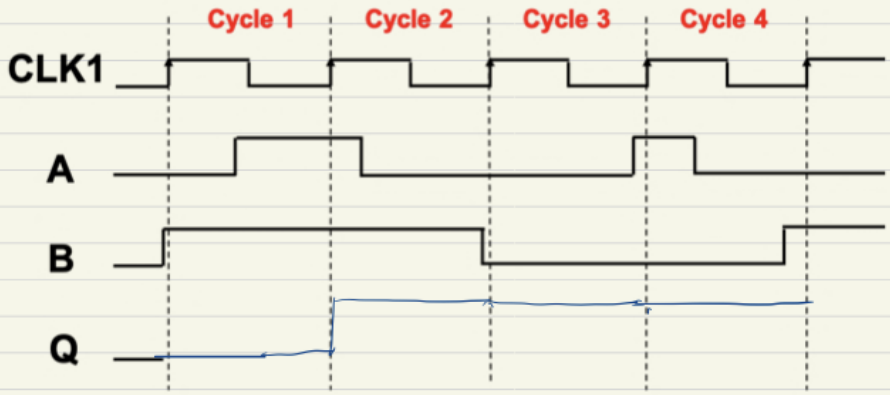


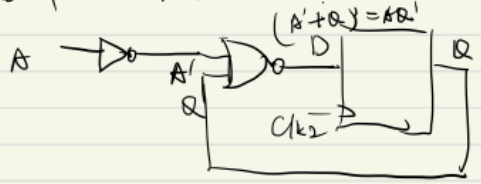
Figure 7(a) Sequential logic FOO

Q7.



a) SET / RESET should be in terms of clk2 $\Rightarrow$ B=1

B=1 $\Rightarrow$ NAND reduced to inverter



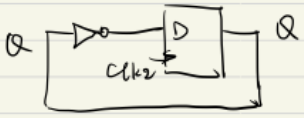
reset: A=0, B=1 $\Rightarrow$ Q=0

Set: unable to set. Need D=1, $\Rightarrow$ Q = A' = 0

The circuit will oscillate. Q 0→1→0.

(After Q=1, D will become 0 again, and Q will become 0)

toggle: A=1, B=1



hold: B=0.

(d) Choose **all** of the following Verilog modules that can correctly implement the FOO circuit. [4 pts]

B, C

Module A

```
module FOO (CLK1, A, B, Q);
  input CLK1, A, B;
  output reg Q;

  wire D, CLK2;
  assign CLK2 = B & CLK1;
  assign D = ~(~(A & B) | Q);

  always @ (CLK2, D)
  begin
    if (CLK2 == 1)
    begin
      Q <= D;
    end
  end
endmodule
```

Module A

Module B

```
module FOO (CLK1, A, B, Q);
  input CLK1, A, B;
  output reg Q;

  wire CLK2;
  assign CLK2 = B & CLK1;

  always @ (posedge CLK2)
  begin
    Q = ~(~(A & B) | Q);
  end
endmodule
```

Module B

Module C

```
module FOO (CLK1, A, B, Q);
  input CLK1, A, B;
  output reg Q;

  wire D, CLK2;
  assign CLK2 = B & CLK1;
  assign D = ~(~(A & B) | Q);

  always @ (posedge CLK2)
  begin
    Q = D;
  end
endmodule
```

Module C

Module D

```
module FOO (CLK1, A, B, Q);
  input CLK1, A, B;
  output reg Q;
  reg D;

  wire CLK2;
  assign CLK2 = B & CLK1;

  always @ (posedge CLK2)
  begin
    D <= ~(A & B);
    Q <= ~(D | Q);
  end
endmodule
```

Module D