Assignment on Architectural Design of Digital Integrated Circuits

Number-12

(Each question carries 10 marks each)

1. A) Negative gate delay - is it possible in the digital circuits or not.

B) Explain Latency and throughput – the two measures of system performance.

A) **Solution**: It seems a bit absurd to have negative value of propagation delay as it provides a misinterpretation of the effect happening before the cause. Common sense says that the output should only change after input. However, under certain special cases, it is possible to have negative delay. In most of such cases, we have one or more of the following conditions:

i) A high drive strength transistor
ii) Slow transition at the input
iii) Small load at the output

Under all of the above mentioned conditions, the output is expected to transition faster than the input signal, and can result in negative propagation delay. An example negative delay scenario is shown in the figure below. The output signal starts to change only after the input signal; however, the faster transition of the output signal causes it to attain 50% level before input signal, thus, resulting in negative propagation delay. In other words, negative delay is a relative concept.

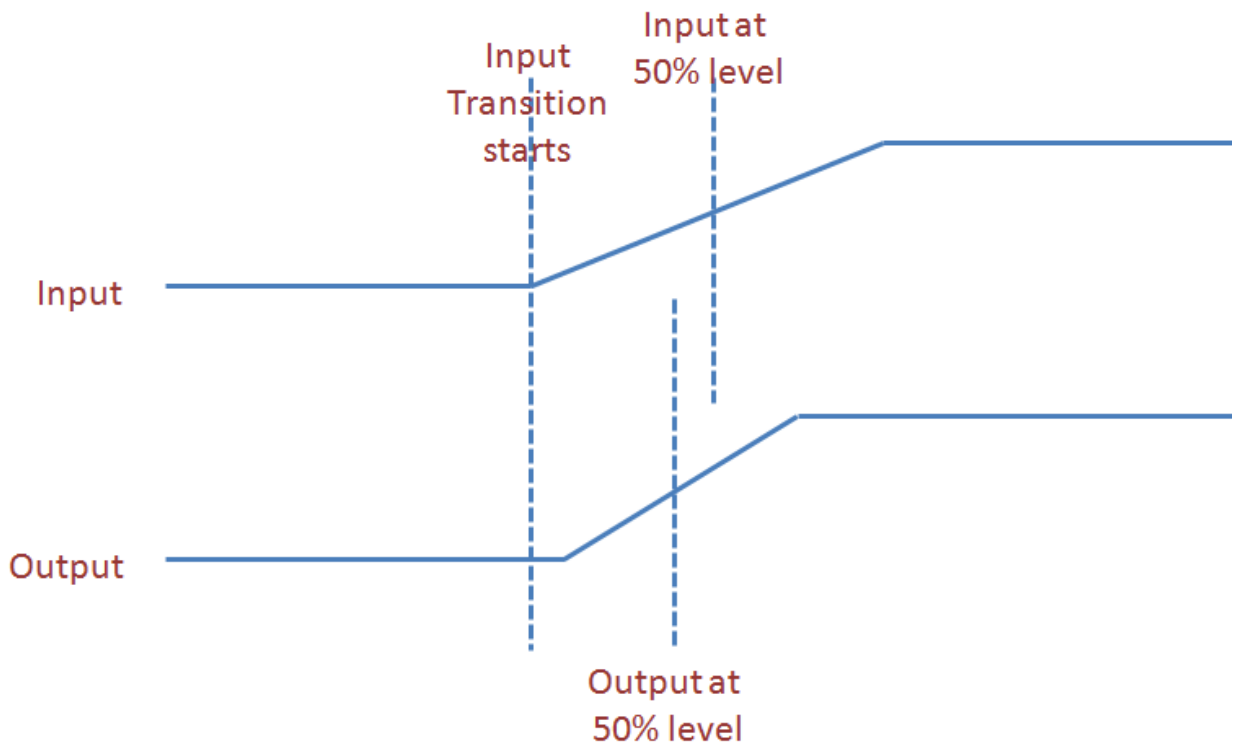
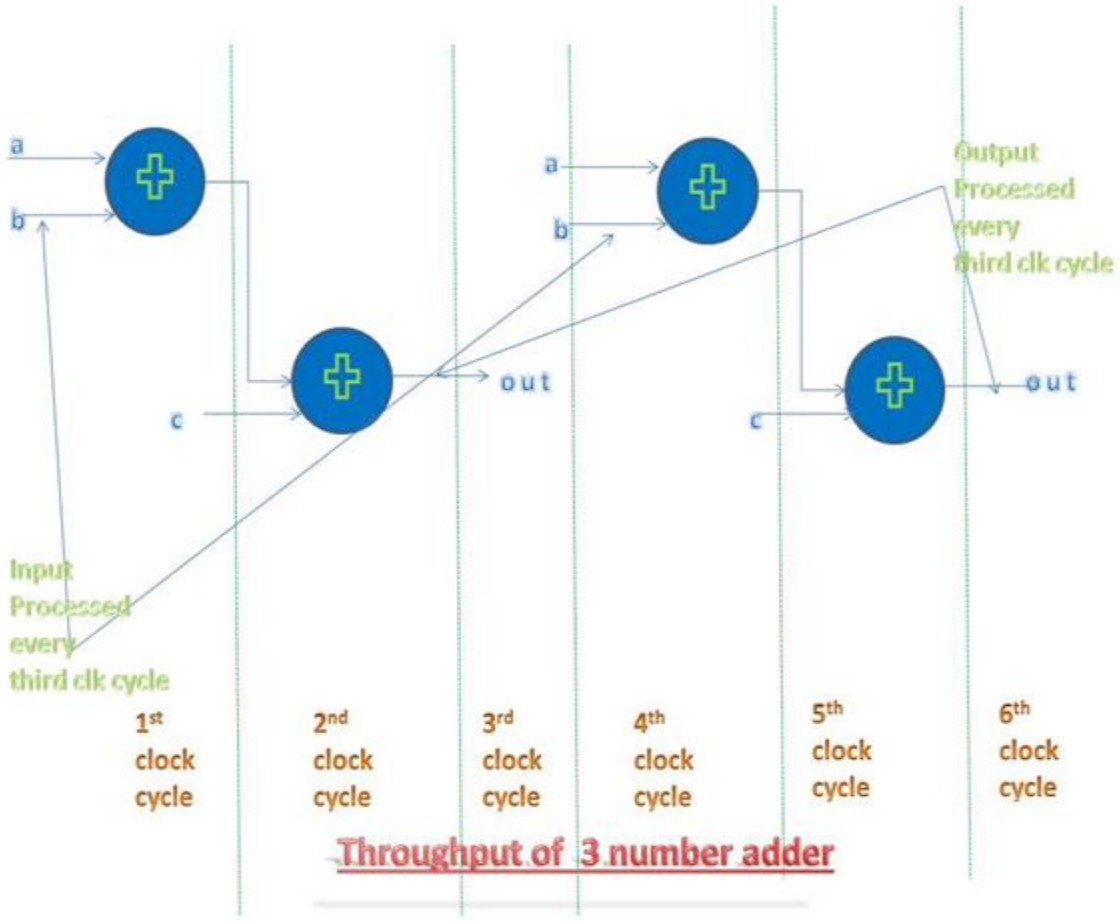


Figure 1: Input and output transitions showing negative input delay

B) Performance of the system is one of the most stringent criteria for its success. While performance increases the desirability among customers, cost is what makes it affordable. This is the reason why system designers aim for maximum performance with available resources such as power and area constraints. There are two related parameters that determine the performance output of a system –

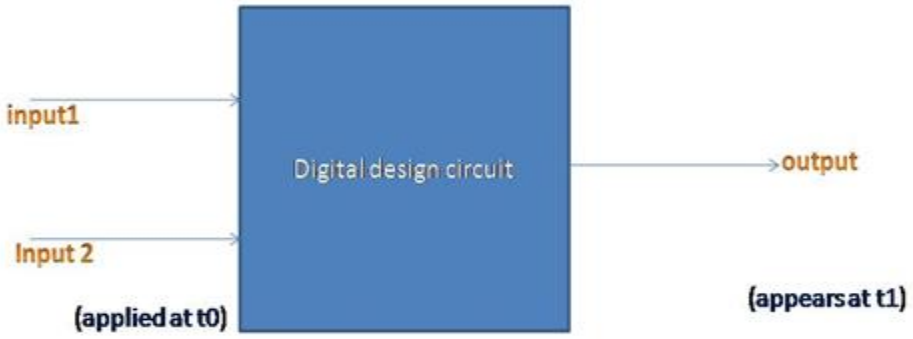
Throughput - Throughput is a measure of the productivity of the system. In electronic/communication systems, throughput refers to rate at which output data is produced. Higher the throughput, more productive is the system. In most of the cases, it is measured as time difference between two consecutive outputs (nth and n+1th). Throughput also refers to the rate at which input data can be applied to system.

Let us discuss with the help of an example:



Above figure depicts the throughput of 3 number adders. Result of input set applied at 1st clock cycle appears at output at 3rd clock cycle and in 4th clock cycle next input set is applied and output comes in 6th clock cycle. Hence, throughput of above design is ⅓ per clock cycle. As we can see from diagram, first input is applied in first clock cycle and 2nd input is applied in 4th clock cycle. Hence we can also say that throughput is rate at which input data can be applied to system.

Latency- Latency is the time taken by a system to produce output after input is applied. It is a measure of delay response of a design. Higher the latency value, slower is the system. in synchronous designs, it is measured in terms of number of clock cycles. In combinational designs, latency is basically propagation delay of circuit. In non pipelined designs, latency improvement is major area of concern. In more general terms, it is time difference between output and input time.



Latency of design = t1 – t0

Relationship between throughput and latency: Both latency and throughput are inter-related. It is desired to have maximum throughput and minimum latency. Increasing latency and/or throughput might make the system costly. Let us take an example. Consider a park with 3 rides and it takes 5 minutes for a ride. A child can take sequentially these rides; i.e, ride 1, ride 2 and then ride 3. Firstly, let us assume that only one child at a time is allowed to enter park at a time. While he is taking a ride, no one is allowed to enter the park. Thus, the throughput of the park is 15 minutes per child and latency is 15 minutes. Now, let us assume that while a child has finished taking ride1, another child is allowed to enter park. Thus, in this case, throughput will be 5 minutes per child whereas latency is still 15 minutes. Thus, we have increased the throughput of the system without affecting latency and at the same cost.

2. Explain need along with the working principle of Reset Synchronizer Solution.

Need for reset synchronizer: The way most of the designs have been modelled needs asynchronous reset assertion and synchronous de-assertion. The requirement of most of the designs these days is:

1. When reset is asserted, it propagates to all designs; brings them to reset state whether or not clock is toggling; i.e. assertion should be asynchronous
2. When reset is deasserted, wait for a clock edge, and then, move the system to next state as per the FSM (Finite State Machine); i.e. deassertion should be synchronous

The top level reset sources are mostly asynchronous, both in assertion and during deassertion. The circuit that manipulates the asynchronous reset to have asynchronous assertion and synchronous deassertion is referred as **reset synchronizer**.

Definition of reset synchronizer: A reset synchronizer synchronizes the deassertion of reset with respect to the clock domain. In other words, a reset synchronizer manipulates the asynchronous reset to have synchronous deassetion.

Figure 1 below shows the schematic representation of how a reset synchronizer is built. It consists of two registers connected in series, the input of first register tied to VDD. The asynchronous reset signal is connected to the **Rbar** pin of both the register. The output of this circuit has synchronized de-assertion. This synchronised reset fans out to the design.

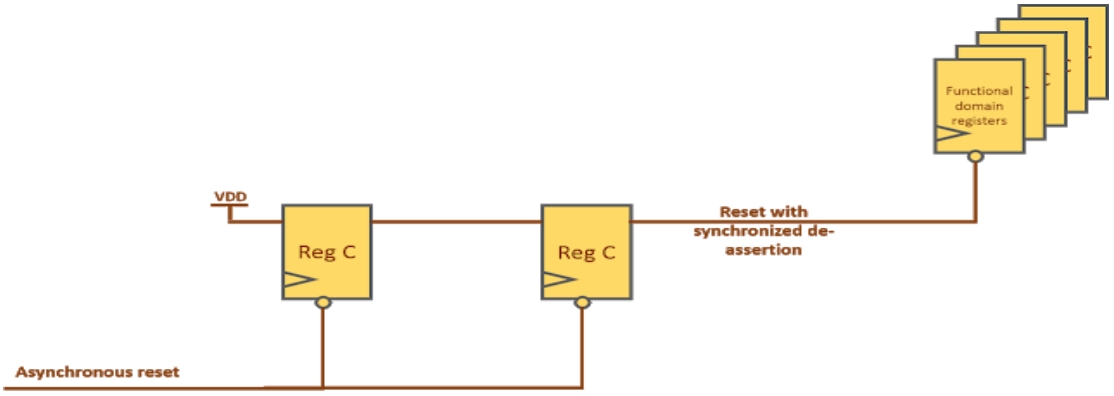


Figure 1: Reset synchronizer

How reset synchronizer works: When the reset is asserted, it first propagates to reset synchronizer flops. It resets both the flops of reset synchronizer asynchronously (without waiting for clock edge) thereby generating reset assertion for fanout registers. Figure 2 below shows the scenario of reset assertion, and also the timing waveforms associated with assertion of reset.

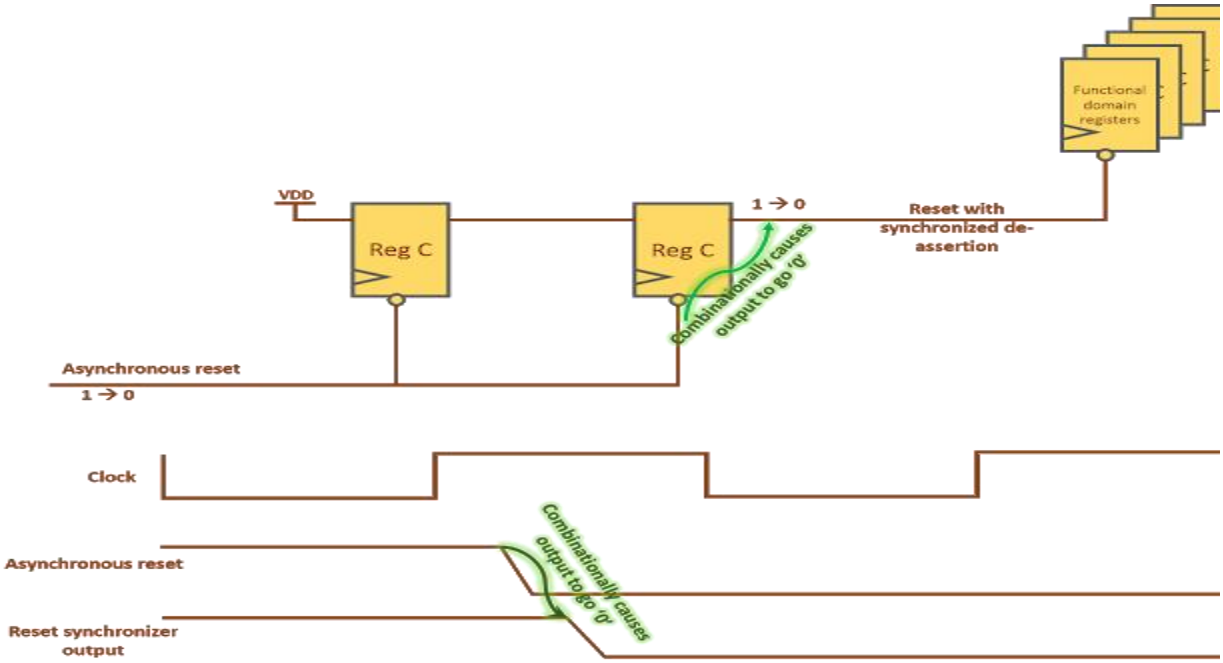


Figure 2: Reset assertion

Similarly, the de-assertion of reset first reaches the two flops of reset synchronizer. Now, the first flop in chain propagates 1 to intermediate output upon arrival of a clock edge. Upon next clock edge, this signal propagates to the output thereby reaching the fanout registers. The reset de-assertion timing (recovery and removal checks timing) should be met from second stage of reset synchronizer to all the domain registers' reset pins as the deassertion is synchronous.

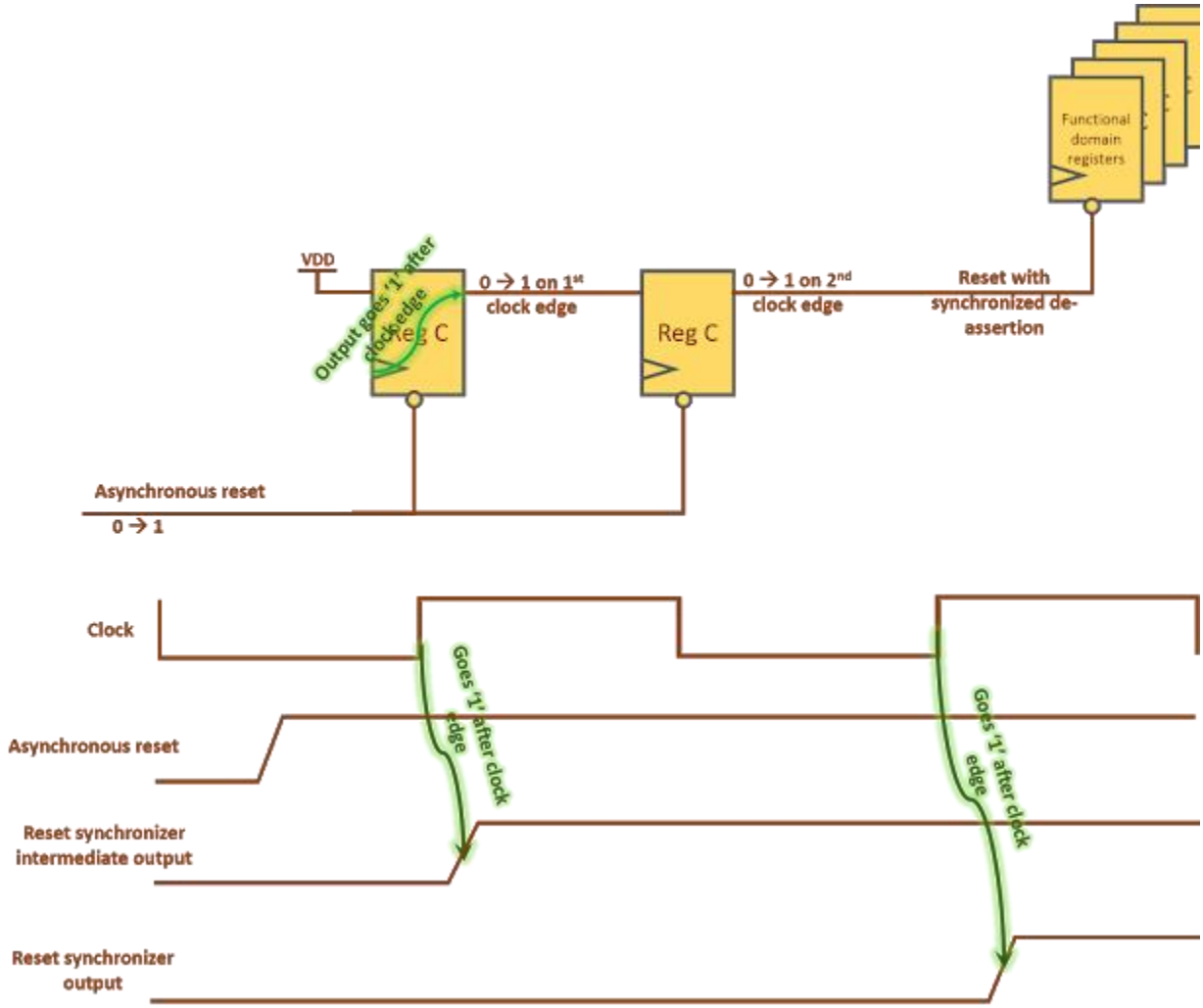


Figure 3: Reset deassertion

Some facts about reset synchronizer:

1. The reset synchronizer manipulates the originally asynchronous reset to have synchronous deassertion.
2. The reset synchronizer must fanout to all the registers that need to be "**OUT OF RESET**" in a single cycle. And there must be a single synchronizer for all such flops, otherwise, some flops will be out of reset 2 cycles later, some 3 cycles later; thus, defeating the purpose of reset synchronization.

3. There can, of course, be multiple reset synchronizers in the design, with the number equal to number of functional clock domains. Each reset synchronizer fans out to all the resettable flops of its own clock domain.

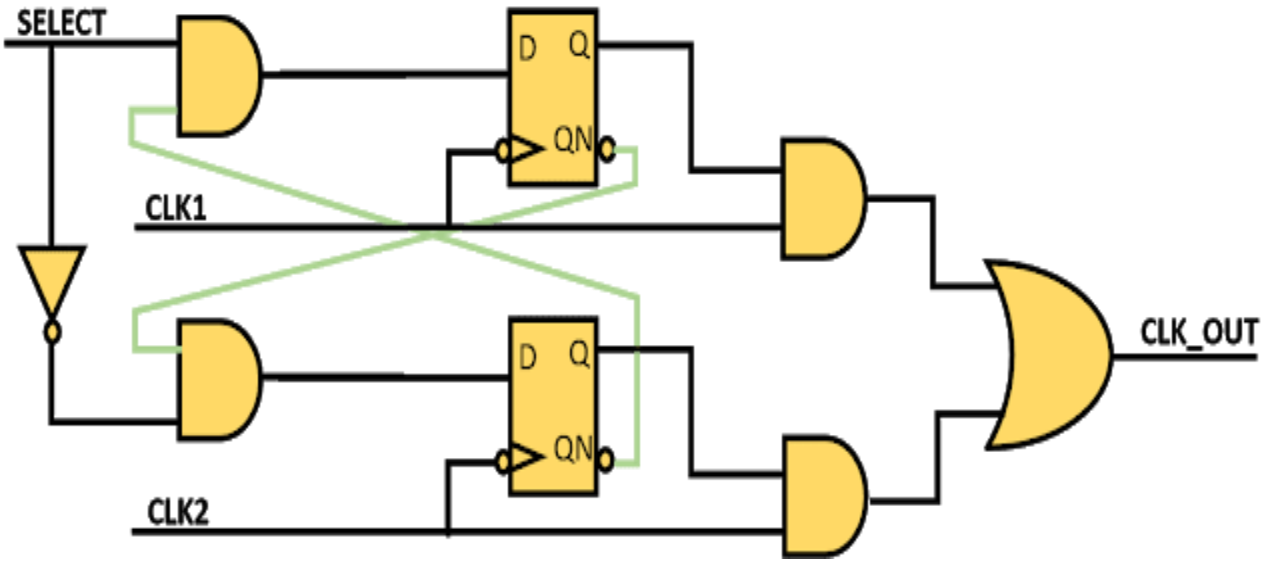
3. A) What is clock multiplexer? Design a circuit for glitch free clock switching between two synchronous clocks.

B) Given an 8:1 multiplexer such that the input connected to 5th input is the most setup timing critical and other inputs are timing critical in the order D0 > D1 > D2 > D3 > D4 > D6 > D7. Restructure the logic accordingly.

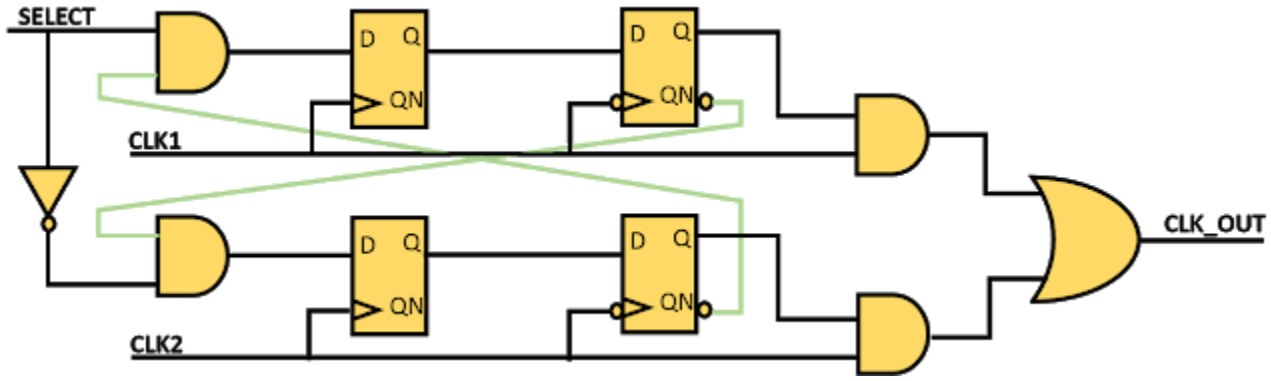
A) **Solution**:

Definition of clock multiplexer: Let us first define a clock multiplexer "A clock multiplexer is a circuit that can switch the system from one clock to another while the chip is running. The two frequencies may be related to each other, or may to totally unrelated". A clock multiplexer switches the clock without any glitches as the glitch in clock will be hazardous for the system. Hence, a clock multiplexer is also known as a glitch-less multiplexer.

Clock multiplexer for switching between two synchronous clocks:

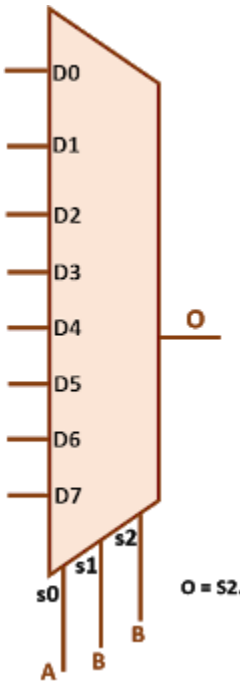


Clock multiplexer for switching between two asynchronous clocks:



B) **Solution**: We know that the most setup timing critical signal should have least logic in the data path. So, we need to prioritize 5th input such that it has least logic out of all the inputs. In other words, this is a problem of converting an ordinary multiplexer to a priority multiplexer. Let us first discuss how we can convert a multiplexer to priority mux.

Figure 1 below shows a multiplexer with 8-inputs D0 - D7 and selects S2,S1,S0.



S2	S1	S0	O
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

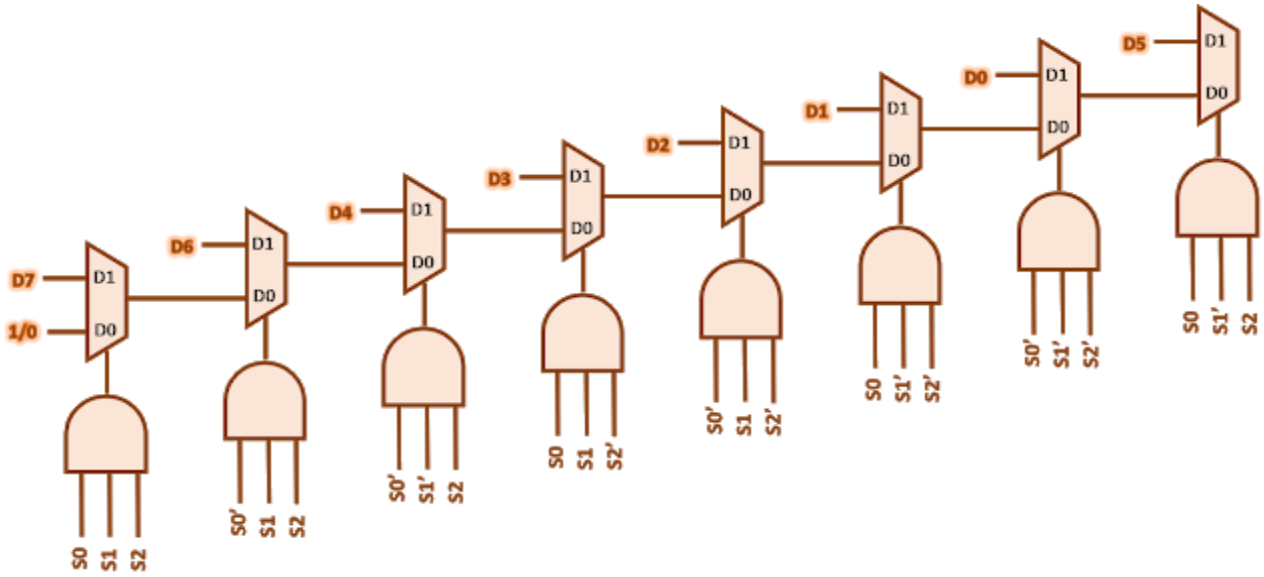
O = S2.S1.S0.D7 + S2.S1.S0'.D6 + S2.S1'.S0.D5 + S2.S1'.S0'.D4 + S2'.S1.S0.D3 + S2'.S1.S0'.D2 + S2'.S1'.S0.D1 + S2'.S1'.S0'.D

Figure 1: 8:1 multiplexer

The equation for output is given as below:

O = S2.S1.S0.D7 + S2.S1.S0'.D6 + S2.S1'.S0.D5 + S2.S1'.S0'.D4 + S2'.S1.S0.D3 + S2'.S1.S0'.D2 + S2'.S1'.S0.D1 + S2'.S1'.S0'.D0

This multiplexer can be represented in the form of a priority multiplexer as required is as shown in figure 2 below.



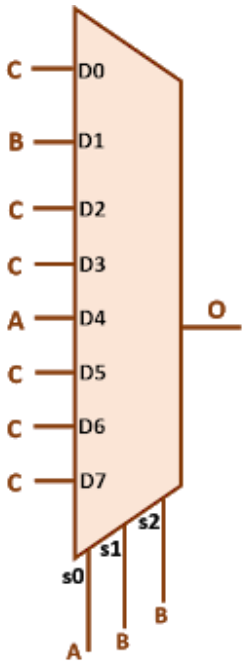
We can start from the equation of the priority multiplexer and prove that it is actually equivalent to 8:1 mux.

The equation of the priority multiplexer is given as:

O = (S0.S1'.S2).D5 + (S0.S1'.S2)'.(S0'.S1'.S2').D0 + (S0.S1'.S2)'.(S0'.S1'.S2').(S0.S1'.S2').D1 + (S0.S1'.S2)'.(S0'.S1'.S2').(S0.S1'.S2')'.(S0'.S1.S2').D2 + (S0.S1'.S2)'.(S0'.S1'.S2').(S0.S1'.S2')'.(S0'.S1.S2')'.(S0.S1.S2').D3 + (S0.S1'.S2)'.(S0'.S1'.S2').(S0.S1'.S2')'.(S0'.S1.S2')'.(S0.S1.S2')'.(S0'.S1.S2).D4 + (S0.S1'.S2)'.(S0'.S1'.S2').(S0.S1'.S2')'.(S0'.S1.S2')'.(S0.S1.S2')'.(S0'.S1.S2).D6 + (S0.S1'.S2)'.(S0'.S1'.S2').(S0.S1'.S2')'.(S0'.S1.S2')'.(S0.S1.S2')'.(S0'.S1.S2).(S0.S1.S2).D7

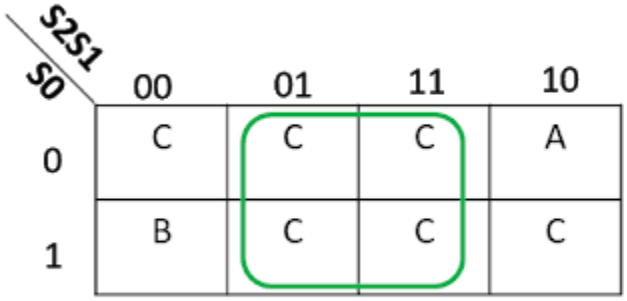
Simplifying the above equation leads us to the equation of ordinary multiplexer.

4. An 8:1 multiplexer selects one out of three inputs based upon different combinations of S2, S1 and S0 as shown in figure below. Minimize the logic with a view that B is the most timing critical input.



S2	S1	S0	O
0	0	0	C
0	0	1	B
0	1	0	C
0	1	1	C
1	0	0	A
1	0	1	C
1	1	0	C
1	1	1	C

Solution: A 4-input MUX has one 4-input AND and one 8-input OR between each input and the output. However, since, there is one signal connected to many of the inputs, there seems to be a scope of logic minimization. Let us use K-map to minimize the logic for problem. The K-map for this problem is as shown below:

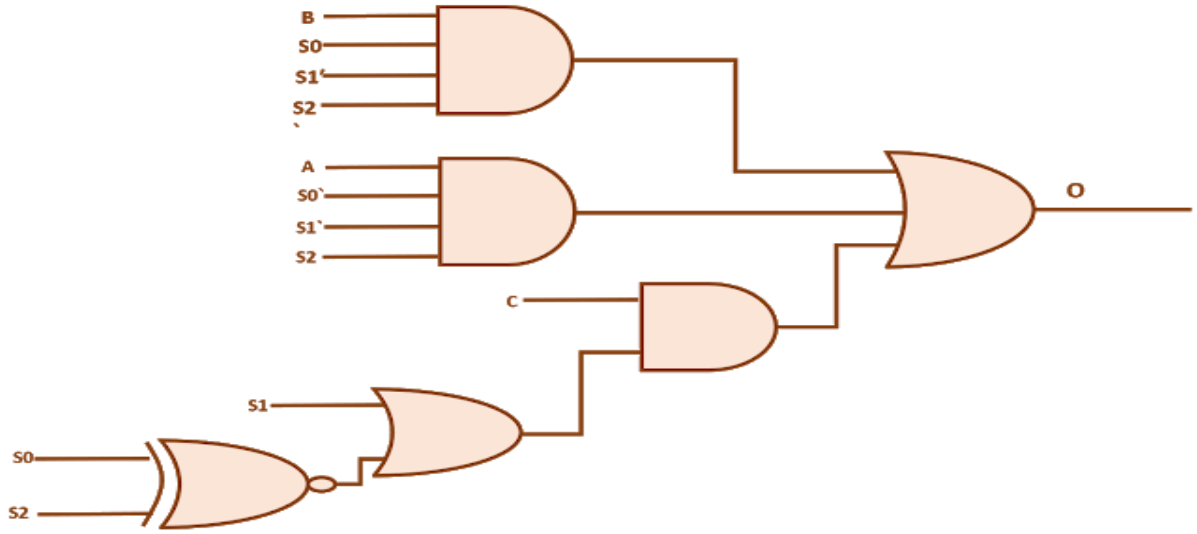


K-map for O(S2,S1,S0,A,B,C)

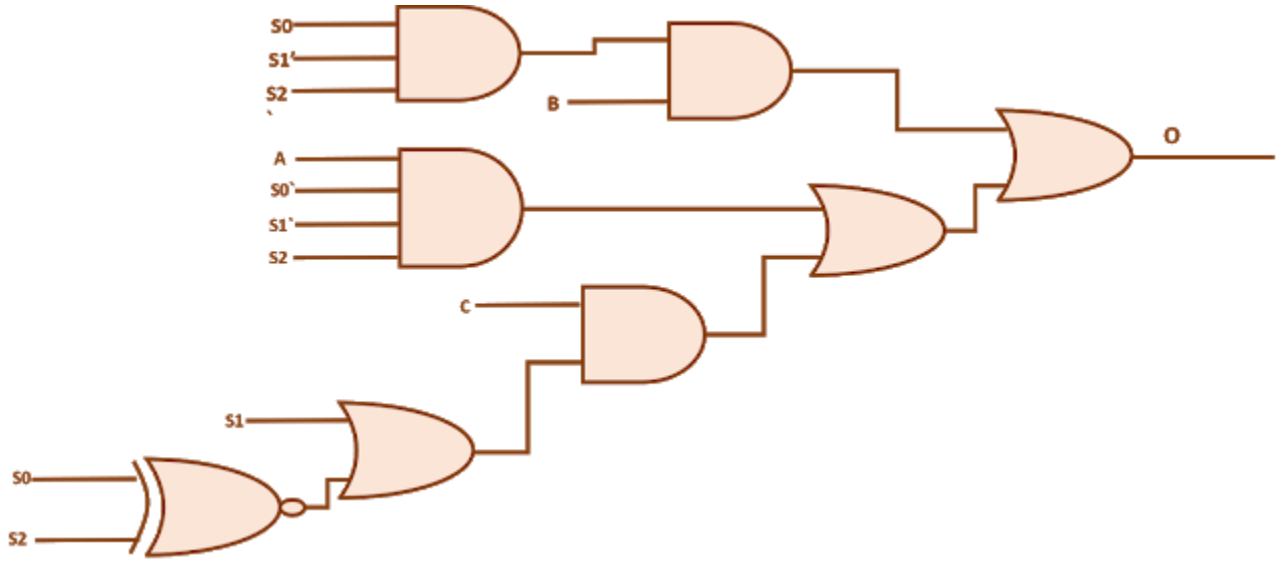
Writing the boolean expression, we get:

O = S2'S1'S0' C + S2'S1'S0 B + S1 C + S2S1'S0'A + S2S1'S0 C
O = C (S1 + S1' (S2 ⊕ S0)) + S2'S1'S0 B + S2 S1' S0' A
O = C (S1 + (S2 ⊕ S0)) + S2'S1'S0 B + S2 S1' S0' A [Using A + A'B = A + B]

If we analyze carefully, we see that **O** is obtained by **OR**-ing three terms; one for A, one for B and one for C. The resulting structure is shown below:

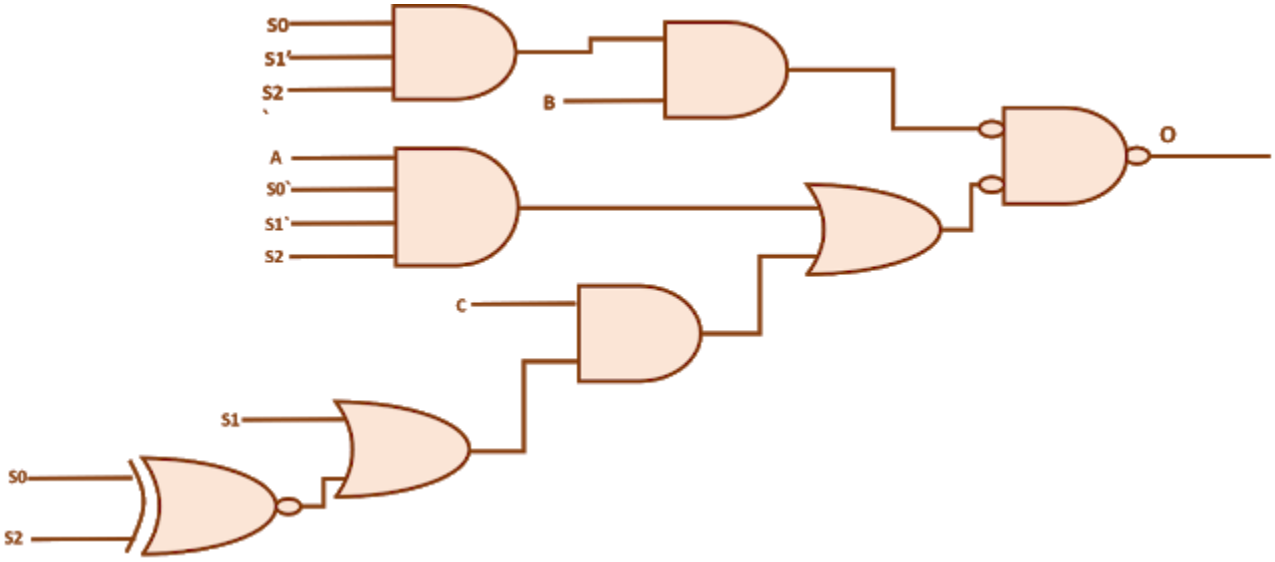


Since, **B** is the most timing-critical input; there should be minimum logic between **B** and output. In other words, it should be closest to output. In the above figure, we see that there is a 4-input AND gate and a 3-input OR gate between **O** and **B**. We can reduce the logic between B and O by breaking 3-input OR into 2-input OR gates such that **B** is closest to output. similarly, we can break the 4-input AND gate into 2-input and 3-input AND gates. Thus, we are left with one 2-input AND gate and one 2-input OR gate between **B** and **O** as shown in figure below.

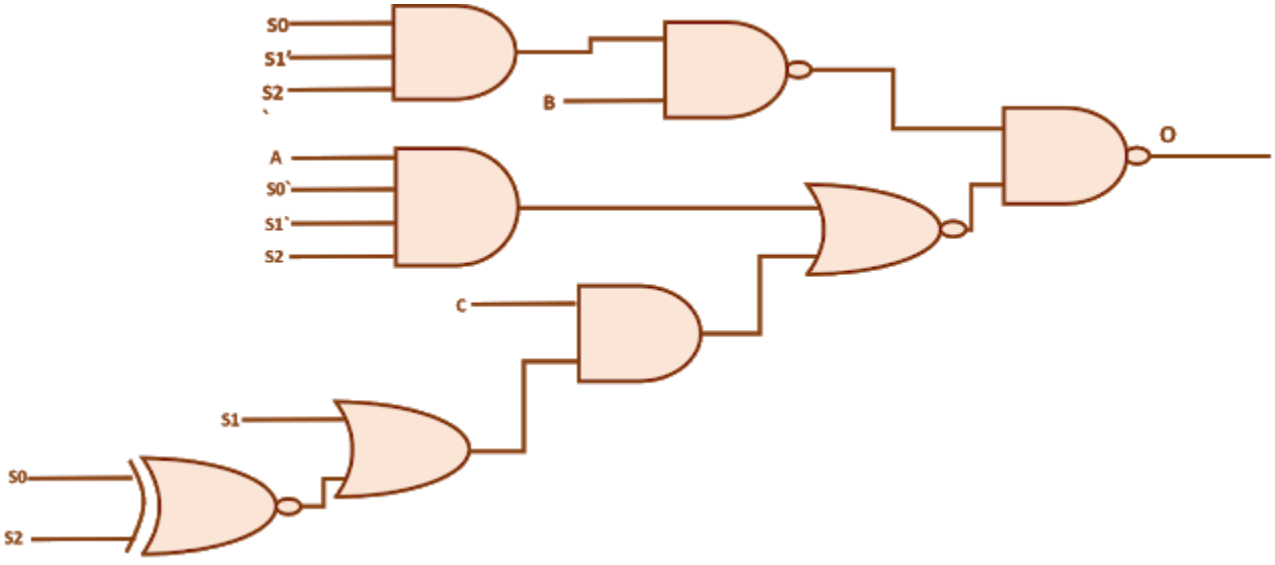


We see that there is still a possibility of logic re-structuring between **B** and **O**. De-Morgans theorem states that
A + B = (A' B')'

Going by this, we can convert the OR gate at the output into NAND gate as shown in figure below.



The bubbles at the input of NAND gate can be moved to the outputs of respective drivers. Or, saying, more sophistically, there are two NAND gates between **B** and **O**.



Thus, we have achieved our purpose of

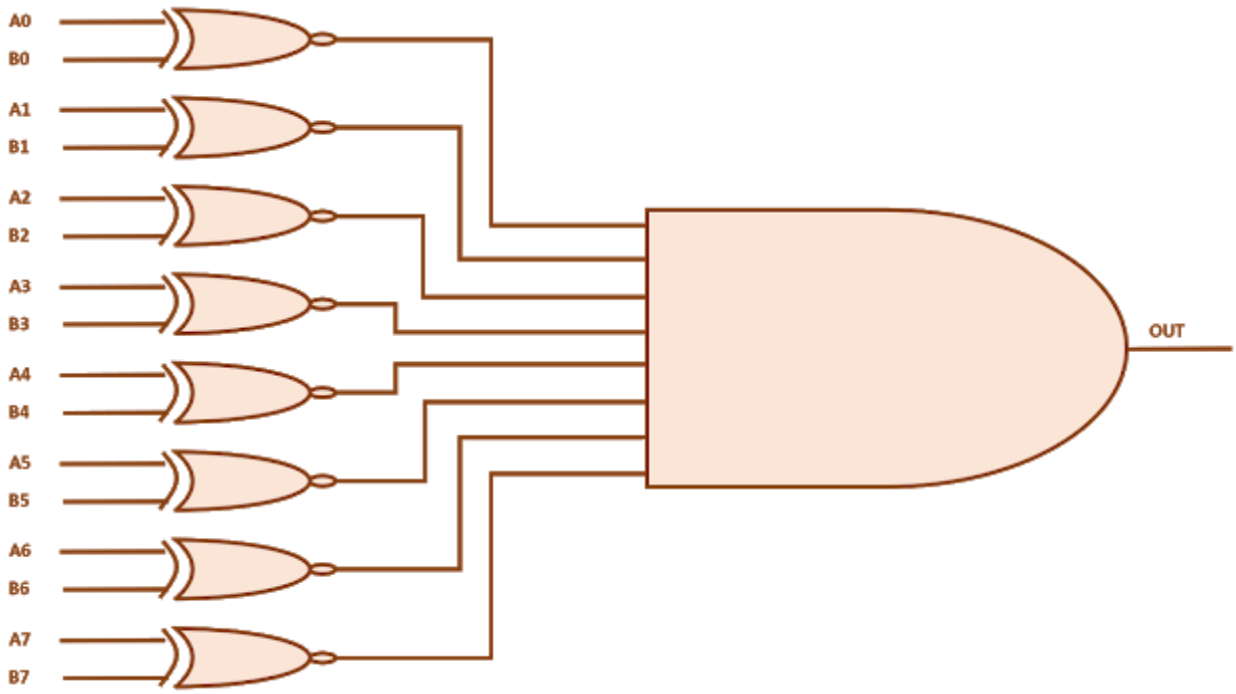
- Minimizing the logic
- Assuring that there is minimum logic between **B** and **O**, since **B** is the most timing critical input.

5. A) How do you detect if two 8-bit numbers/signals are equal?

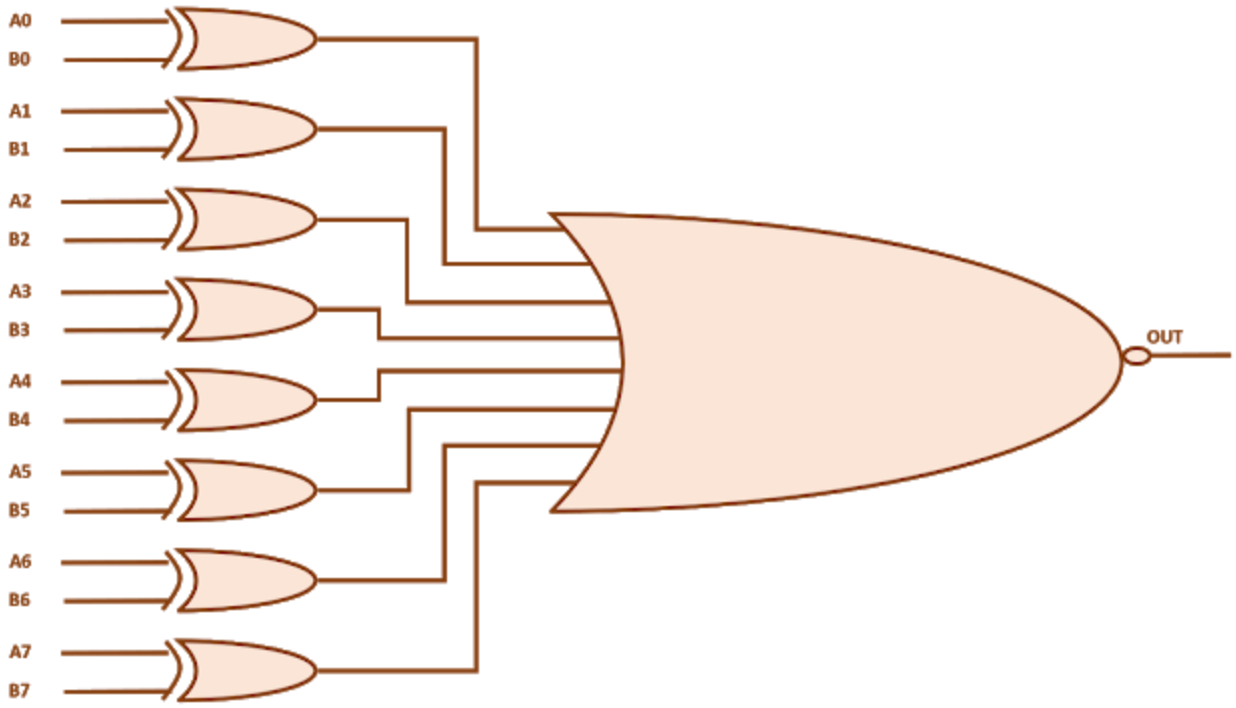
B) Design a circuit that delays the positive edge of a signal by one cycle.

A) Solution:

Here, the problem involves detecting if each bit of a signal is equal to corresponding bit of the other signal and then generating a resultant. First of all, the circuit which provides equivalence of 1-bit is nothing but an XNOR gate. So, we require 8 XNOR gates to judge equivalence of individual bits. Even if one of the bits is "0", it means the numbers are not equal, which can be obtained by ANDing the eight bits together.



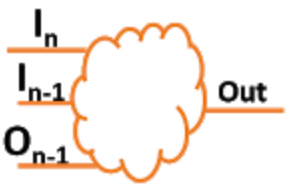
Alternatively, we can use an XOR gate as well. An XOR gate provides output as "1" if the two inputs are not equal. Even if one of the 8 individual XOR gates provides output as "1", it will mean that the numbers are not equal, which can be obtained by NORing the eight bits together.



B) **Solution**: Here, we are given a problem wherein only (0 -> 1) transition of the signal is delayed by a single clock cycle whereas the other transition changes the output combination-ally. In other words, we are given the task to implement a Mealy state machine as output is both a function of state variables and input. However, this is a pretty simple problem involving single state. The output is a function of:

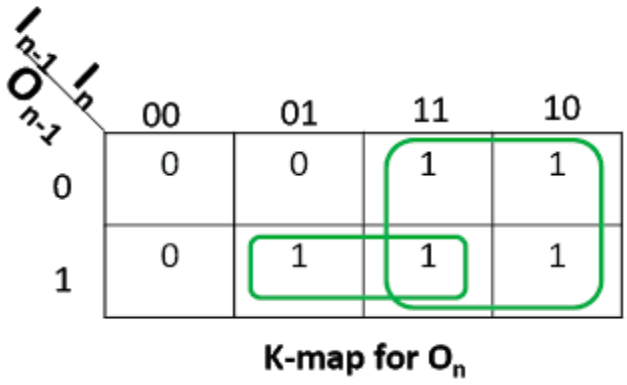
- Present input
- Input value one cycle before

Output should go "0" as soon as input goes "0". But it should go "1" when input one cycle back is "1". But there is a twist. What if current input is "0" and one cycle back, it was "1"? There is no clarity in the problem statement. Let us assume the output remains unchanged in such condition. The state transition table looks as shown below:



I_{n-1}	I_n	O_{n-1}	O_n
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

We can use K-map to solve for O. The solution is given in the figure below:



K-map for O_n

$$O_n = I_{n-1} + I_n O_{n-1}$$

The resulting circuit is as shown in figure below.