Assignment on Architectural Design of Digital Integrated Circuits

Number-11

(Each question carries 10 marks each)

1. A) Draw the circuit of a 4x1 mux using 2 input NAND gates only.

B) Draw the circuit of a 4x1 mux using any input NAND gates.

Solution: Here, we will discuss how we can use NAND gates to build a 4x1 mux:

1. **Using structural approach**: As we know that a 4x1 mux can be structurally built from 2x1 muxes as shown in figure 1 below. Thus, in the same way, we can arrange the 2-input NAND gates to build 4x1 muxes as shown in figure 1.

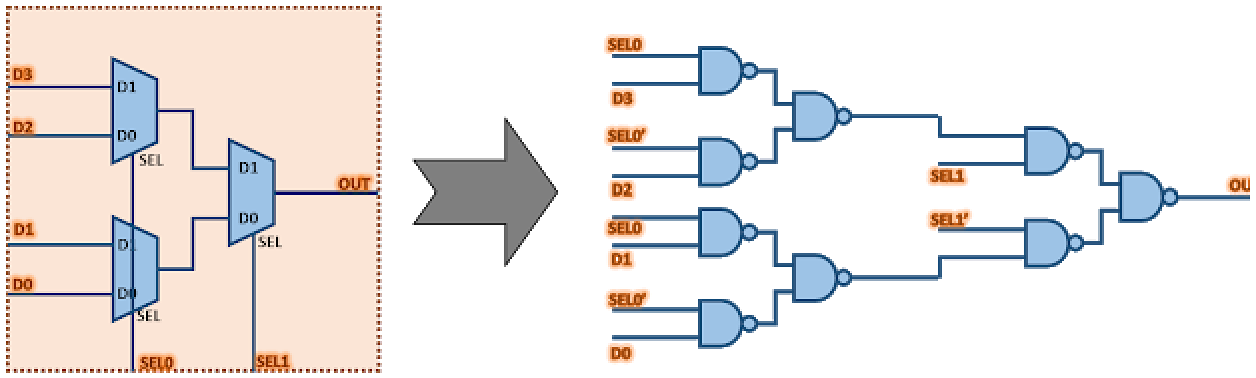


Figure 1: 4x1 mux using NAND gates with structural approach

2. **Building 4x1 mux directly from NAND gates**: The logical equation of a 4x1 multiplexer is given as:
Y = (S1' S0' A + S1' S0 B + S1 S0' C + S1 S0 D)
where S1 and S0 are the selects of the multiplexer and A, B, C and D are the multiplexer inputs.

Now, using De-morgan's law (**m + n = (m'n')'**)

The above equation turns into,
Y = ((S1' S0' A)' (S1' S0 B)' (S1 S0' C)' (S1 S0 D)')'
In other words,

Y = NAND (NAND(S1',S0',A),NAND(S1',S0,B),NAND(S1,S0',C),NAND(S1,S0,D))
Thus, we require four 3-input NAND gates and a 4-input NAND gate to implement a 4x1 mux. The implementation is shown in figure 2 below.

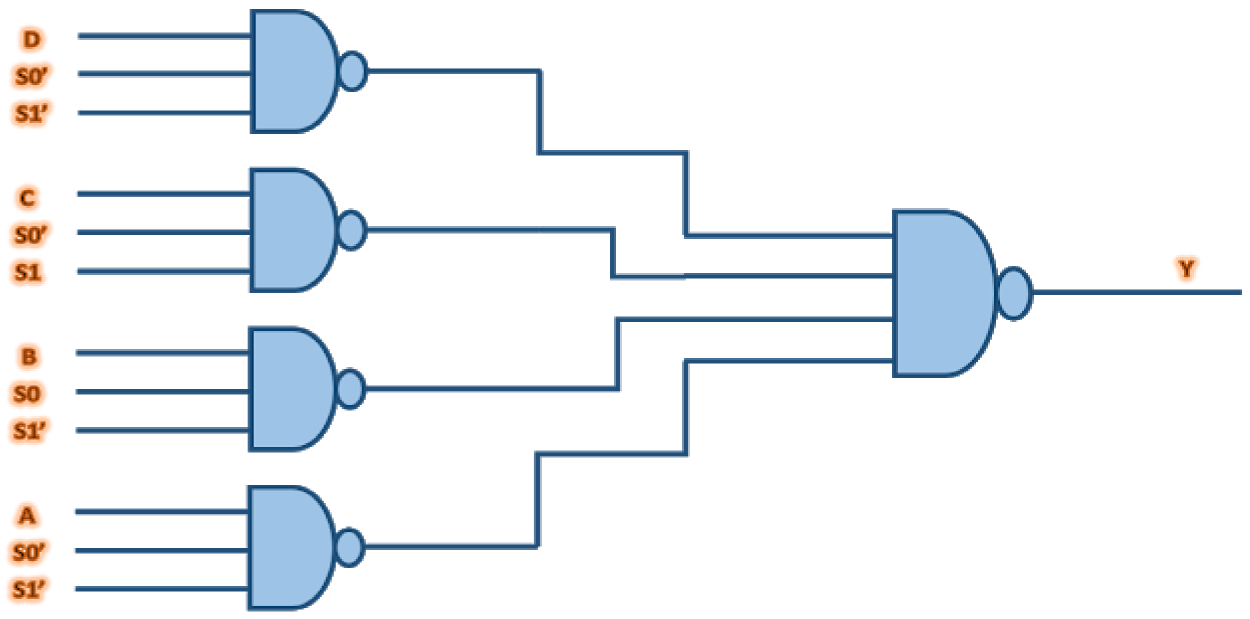
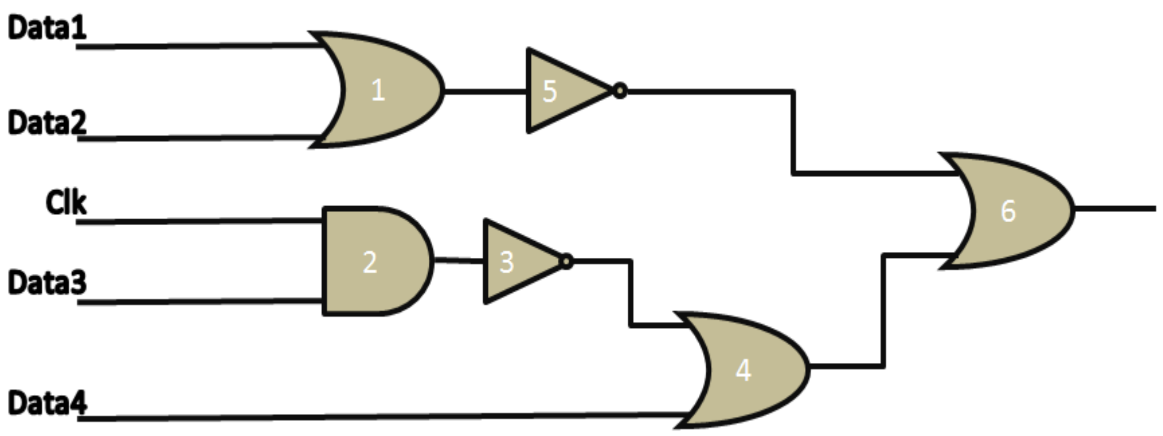


Figure 2: 4x1 mux using NAND gates

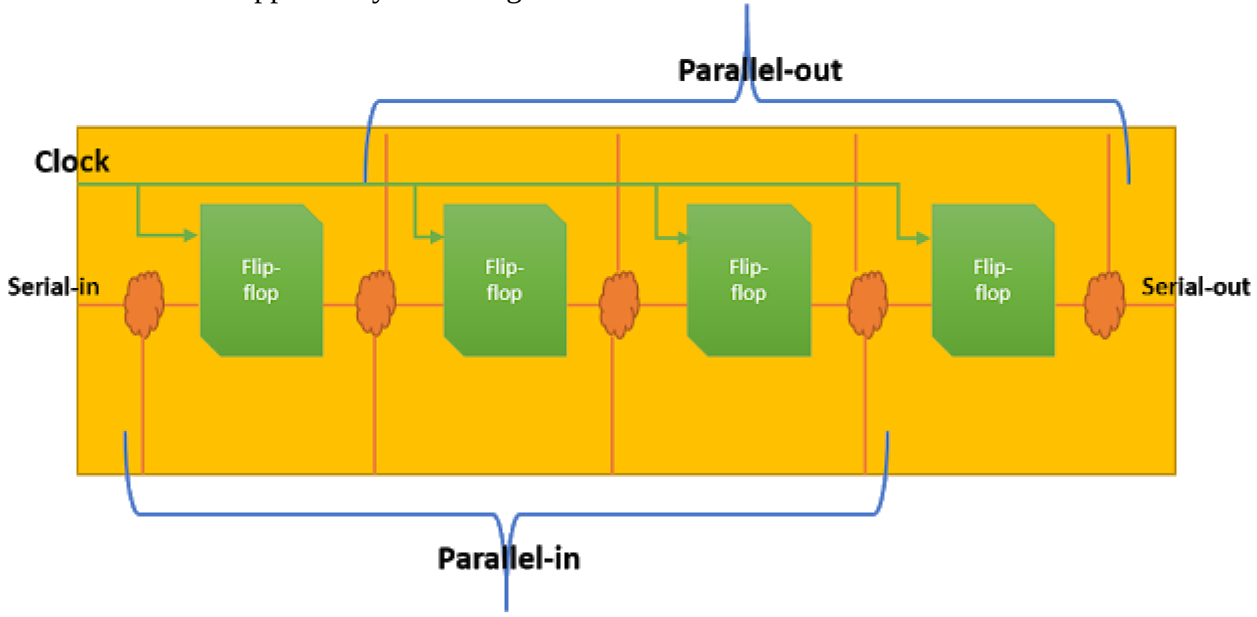
2. A) Draw the Clock gated version for a 4-bit shift register.

B) Consider a complex gate with internal structure as shown in figure below. One of the inputs gets clock while all others get data signals. What all (and what type of) clock gating checks exist?



A) Solution:

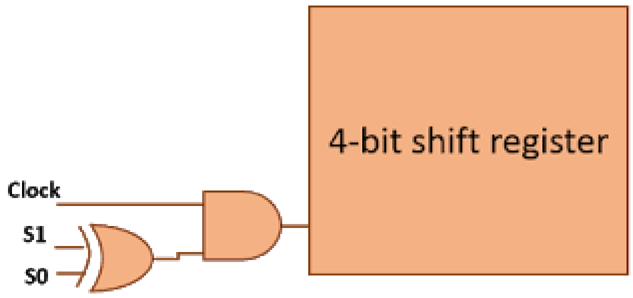
There is an 4-bit shift register with parallel read and write capability as shown in the diagram. We need to find out an opportunity to clock gate the module.



Mode selection bits ("S1" and "S0") are controlling the operation of this shift register with following settings:

Mode selection bits	Operation	State of flip-flops	Serial-out	Parallel-out
00	No operation	No change	0	0000
01	Serial-shift	Data shifts	F4	0000
10	Parallel load	Parallel-in loaded	0	0000
11	Parallel read	No change	0	F1,F2,F3,F4

From the basics of clock gating, we know that if the state of a flip-flop is not changing, there lies an opportunity to gate its clock. Observing the table, we see that state of all flip-flops does not change when "S1, S0" are either "00" or "11". So, when mode selection bits are corresponding to these values, we can gate the clock to this shift register. Or, we can say that clock to the module should reach only when (S1 xor S0) is equal to 1.



B) **Solution:** As we know, clock gating checks can be of AND type or OR type. We can find the type of clock gating check formed between a data and a clock signal by considering all other signals as constant. Since, all the 4 data signals control **Clk** in one or the other way, there are following clock gating checks formed:

i) **Clock gating check between Data1 and Clk:** As is evident, invert of **Clk** and **Data1** meet at OR gate '6'. Hence, there is OR type check between **invert of Clk** and **Data1.** In other words, **Data1** can change only when **invert of Clk is high** or **Clk is low**. Hence, there is **AND type check formed at gate 6**.

ii) **Clock gating check between Data2 and Clk:** Same as in case 1.

iii) **Clock gating check between Data3 and Clk**: There is **AND type check between Data3 and Clk.**

iv) **Clock gating check between Data4 and CLK**: As in 1 and 2, there is **AND type check between Data4 and Clk**.

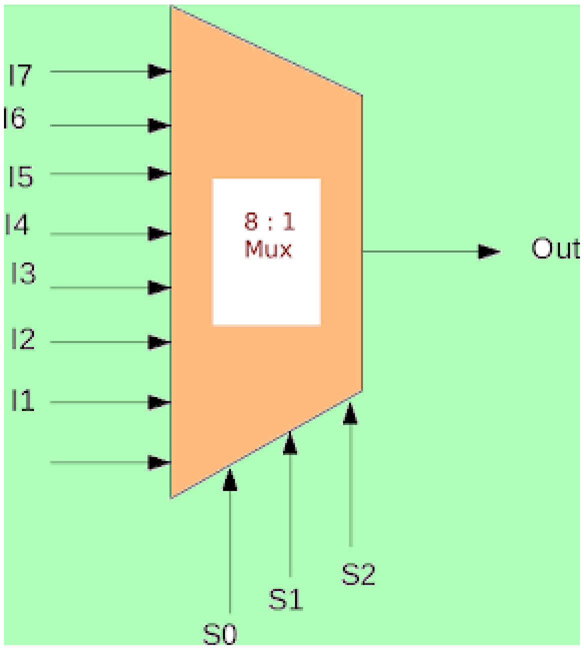
3. Implement 3 and 4 variable functions using 8:1 MUX Implement 3 and 4 variable functions using 8:1 MUX?

Solution:

Three variable functions can be easily implemented using 8:1 multiplexer. Connect 3 input lines to select lines of mux and connect 8 inputs of mux to logic 0 or 1 according to function output. For example, let us say Function is

F(X,Y,Z) = Σ(0,1,3,6)

then X,Y,Z will be connected to select lines of Mux and I0 , I1, I3 and I6 will be connected to logic 1(VDD) and other will be connected to logic 0

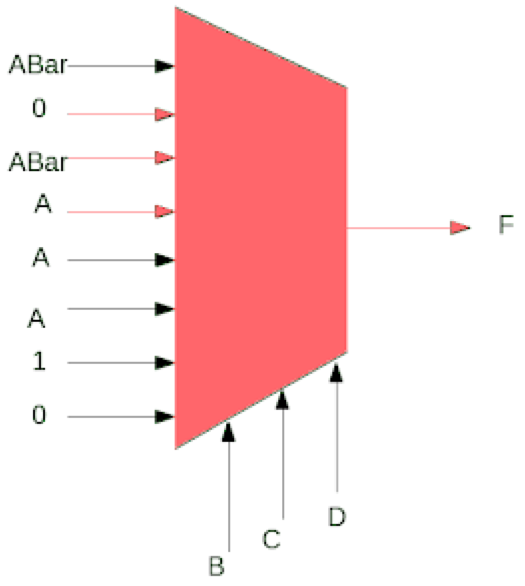


For a 4 variable function, there are 16 possible combinations. To implement 4 variable functions using 8:1 MUX, use 3 inputs as select lines of MUX and remaining 4th input and function will determine ith input of mux. Let us demonstrate it with an example:

F(A,B,C,D) = Σ(1,5,7,9,10,11,12)

A	B	C	D	Decimal Equivalent	F
0	0	0	0	0	0
1	0	0	0	8	0
0	0	0	1	1	1
1	0	0	1	9	1
0	0	1	0	2	0
1	0	1	0	10	1
0	0	1	1	3	0
1	0	1	1	11	1

0	1	0	0	4	0
1	1	0	0	12	1
0	1	0	1	5	1
1	1	0	1	13	0
0	1	1	0	6	0
1	1	1	0	14	0
0	1	1	1	7	1
1	1	1	1	15	0



The 4 variable function represented using 8:1 mux

ABar = ~A (inverted A)

As shown in figure, B, C, D is used as select lines and A will be used input of Mux. from Truth table, if B,C,D are 0 then output F is 0 irrespective of status of A so I0 = 0. For I5 (BCD = 101) output depend upon A.

for A = 0, F = 1
for A = 1, F = 0
Hence F = ~A (for BCD = 101)
I4, (B = 1, C = 0, D = 0), F = A
I1, (B = 0, C = 0,D = 1), F = 1 (irrespective of status of A)

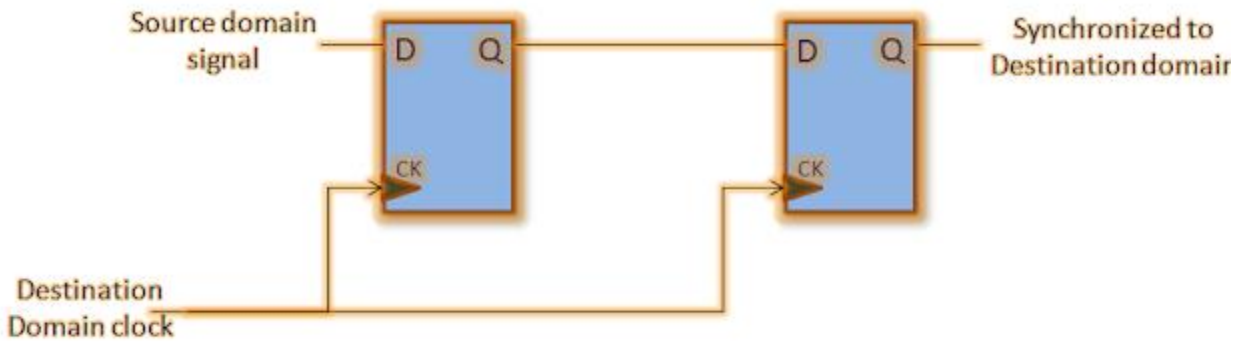
Similarly all other inputs can be inferred in the same way. Thus we can conclude that to implement n variable function, we need **2^(n-1) to 1 MUX** and an **inverter**. n-1 input lines shall be used as select lines and rest one will be used for input of MUX.

4. Explain the need for synchronizer in modern days VLSI circuit design? What type of synchronizers is mainly used VLSI circuit design. Draw the logic design of them with their working principle.

Solution: Modern VLSI designs have very complex architectures and multiple clock sources. Multiple clock domains interact within the chip. Also, there are interfaces that connect the chip to the outside world. If these different clock domains are not properly synchronized, metastability events are bound to happen and may result in chip failure. Synchronizers come to rescue to avoid fatal effects of metastability arising due to signals crossing clock domain boundaries and are must where two clock domains interact. Understanding metasbility and correct design of synchronizers to prevent metastability happen is an art. For systems with only one clock domain, synchronizers are required only when reading an asynchronous signal.

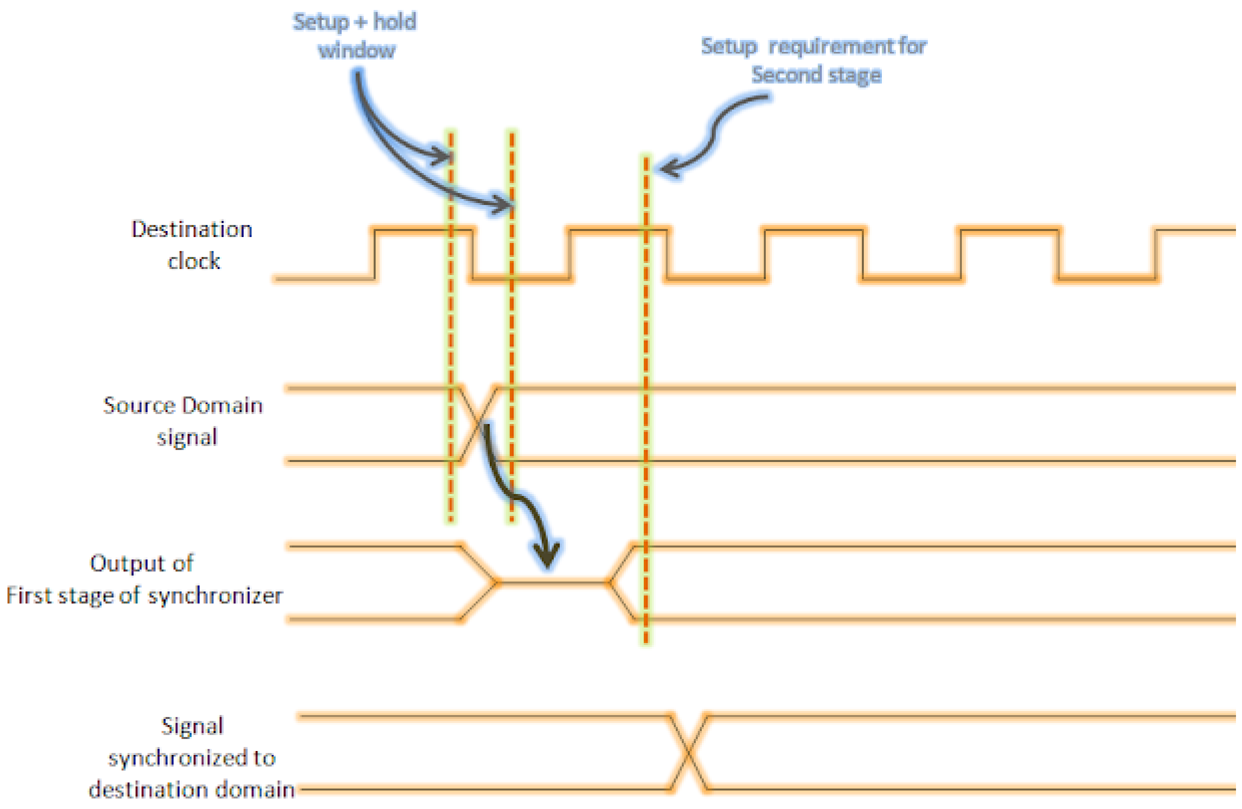
Synchronizers, a key to tolerate metastability: As mentioned earlier, asynchronous signals cause catastrophic metastability failures when introduced into a clock domain. So, the first thing that arises in one's mind is to find ways to avoid metastability failures. But the fact is that metastability cannot be avoided. We have to learn to tolerate metastability. This is where synchronizers come to rescue. A synchronizer is a digital circuit that converts an asynchronous signal/a signal from a different clock domain into the recipient clock domain so that it can be captured without introducing any metastability failure. However, the introduction of synchronizers does not totally guarantee prevention of metastability. It only reduces the chances of metastability by a huge factor. Thus, a good synchronizer must be reliable with high MTBF, should have low latency from source to destination domain and should have low area/power impact. When a signal is passed from one clock domain to another clock domain, the circuit that receives the signal needs to synchronize it. Whatever metastability effects are to be caused due to this, have to be absorbed in synchronizer circuit only. Thus, the purpose of synchronizer is to prevent the downstream logic from metastable state of first flip-flop in new clock domain.

Flip-flop based synchronizer (Two flip-flop synchronizer): This is the most simple and most common synchronization scheme and consists of two or more flip-flops in chain working on the destination clock domain. This approach allows for an entire clock period for the first flop to resolve metastability. Let us consider the simplest case of a flip-flop synchronizer with 2 flops as shown in figure. Here, Q2 goes high 1 or 2 cycles later than the input.



A two flip-flop synchronizer

As said earlier, the two flop synchronizer converts a signal from source clock domain to destination clock domain. The input to the first stage is asynchronous to the destination clock. So, the output of first stage (Q1) might go metastable from time to time. However, as long as metastability is resolved before next clock edge, the output of second stage (Q2) should have valid logic levels (as shown in figure 3). Thus, the asynchronous signal is synchronized with a maximum latency of 2 clock cycles.Theoretically, it is still possible for the output of first stage to be unresolved before it is to be sampled by second stage. In that case, output of second stage will also go metastable. If the probability of this event is high, then you need to consider having a three stage synchronizer.


Setup + hold window
Setup requirement for Second stage
Destination clock
Source Domain signal
Output of First stage of synchronizer
Signal synchronized to destination domain

Waveforms in a two flop synchronizer

A good two flip-flop synchronizer design should have following characteristics:

- The two flops should be placed as close as possible to allow the metastability at first stage output maximum time to get resolved. Therefore, some ASIC libraries have built in synchronizer stages. These have better MTBF and have very large flops used, hence, consume more power.

- Two flop synchronizer is the most basic design all other synchronizers are based upon.

- Source domain signal is expected to remain stable for minimum two destination clock cycles so that first stage is guaranteed to sample it on second clock edge. In some cases, it is not possible even to predict the destination domain frequency. In such cases, handshaking mechanism may be used.

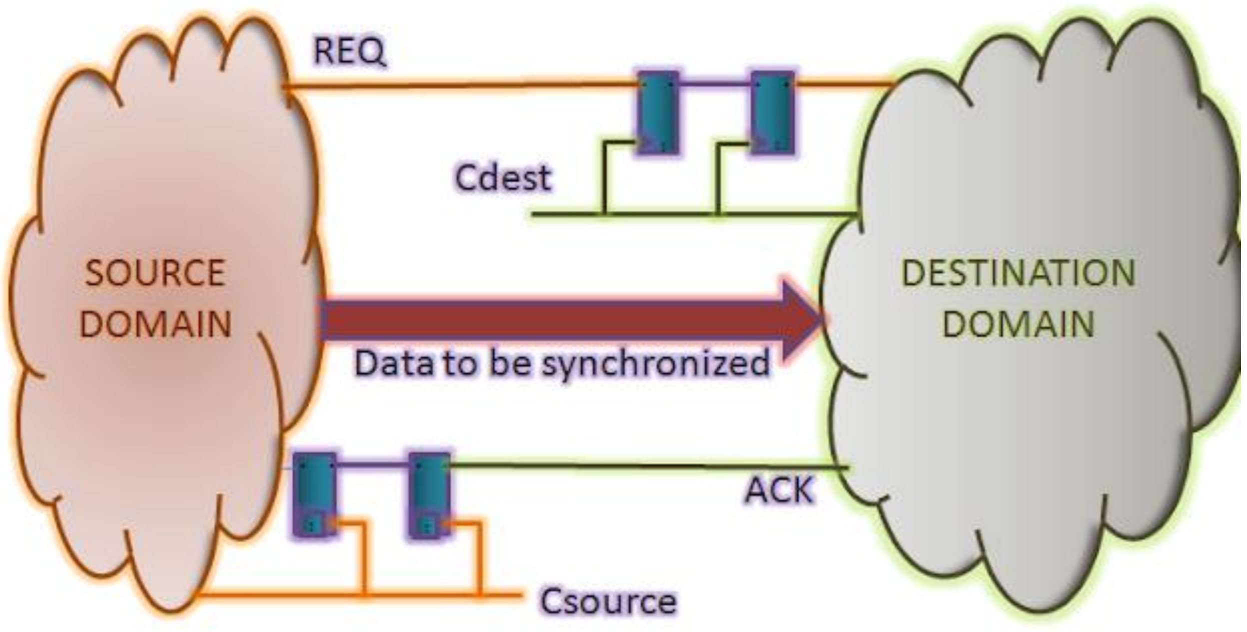
- The two flop synchronizer must be used to synchronize a single bit data only. Using multiple two flops synchronizers to synchronize multi-bit data may lead to catastrophic results as some bits might pass through in first cycle; others in second cycle. Thus, the destination domain FSM may go in some undesired state.

- Another practice that is forbidden is to synchronize same bit by two different synchronizers. This may lead to one of these becoming 0, other becoming 1 leading into inconsistent state.

- The two stages in flop synchronizers are not enough for very high speed clocks as MTBF becomes significantly low (eg. in processors, where clocks run in excess of 1 GHz). In such cases, adding one extra stage will help.

- MTBF decreases almost linearly with the number of synchronizers in the system. Thus, if your system uses 1000 synchronizers, each of these must be designed with atleast 1000 times more MTBF than the actual reliability target.

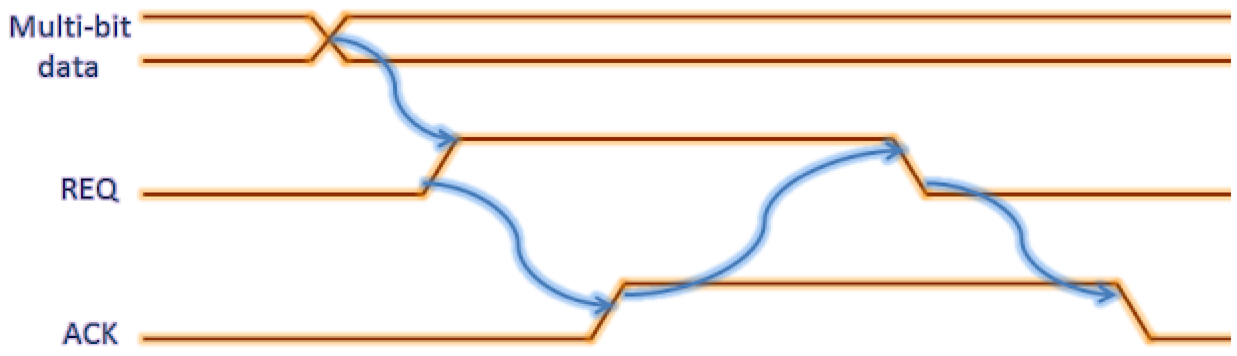
Handshaking based synchronizers: As discussed earlier, two flop synchronizer works only when there is one bit of data transfer between the two clock domains and the result of using multiple two-flop synchronizers to synchronize multi-bit data is catastrophic. The solution for this is to implement handshaking based synchronization where the transfer of data is controlled by handshaking protocol wherein source domain places data on the ‘REQ’ signal. When it goes high, receiver knows data is stable on bus and it is safe to sample the data. After sampling, the receiver asserts ‘ACK’ signal. This signal is synchronized to the source domain and informs the sender that data has been sampled successfully and it may send a new data. Handshaking based synchronizers offer a good reliable communication but reduce data transmission bandwidth as it takes many cycles to exchange handshaking signals. Handshaking allows digital circuits to effectively communicate with each other when response time of one or both circuits is unpredictable.



Handshaking protocol based synchronization technique

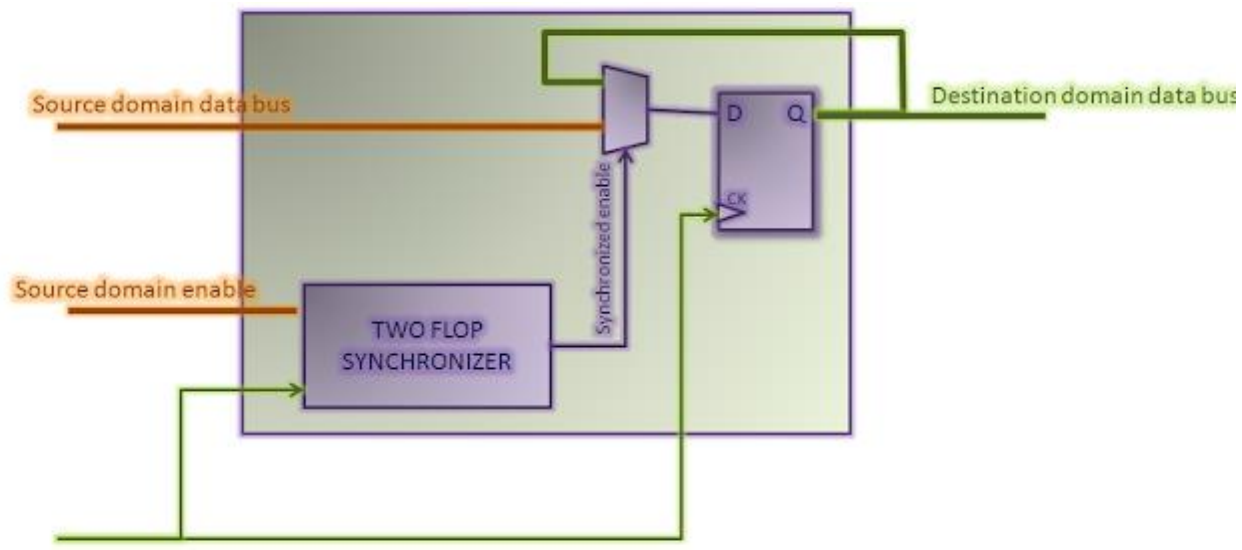
How handshaking takes place:

1.) Sender places data, then asserts REQ signal
2.) Receiver latches data and asserts ACK
3.) Sender deasserts REQ
4.) Receiver deasserts ACK to inform the sender that it is ready to accept another data. Sender may start the handshaking protocol again.



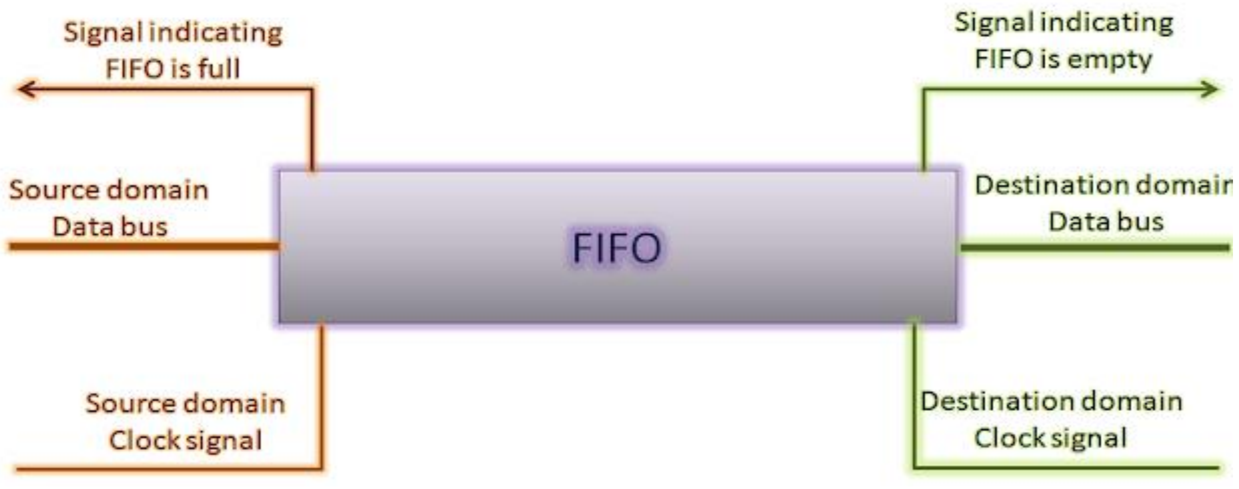
Sequence of events in a handshaking protocol based synchronizer

Mux based synchronizers: As mentioned above, two flop synchronizers are hazardous if used to synchronize data which is more than 1-bit in width. In such situations, we may use mux-based synchronization scheme. In this, the source domain sends an enable signal indicating that it the data has been changed. This enable is synchronized to the destination domain using the two flop synchronizer. This synchronized signal acts as an enable signal indicating that the data on data bus from source is stable and destination domain may latch the data. As shown in the figure, two flop synchronizer acts as a sub-unit of mux based synchronization scheme.



A mux-based synchronization scheme

<u>Two clock FIFO synchronizer</u>: FIFO synchronizers are the most common fast synchronizers used in the VLSI industry. There is a '**cyclic buffer**' (dual port RAM) that is written into by the data coming from the source domain and read by the destination domain. There are two pointers maintained; one corresponding to write, other pointing to read. These pointers are used by these two domains to conclude whether the FIFO is empty or full. For doing this, the two pointers (from different clock domains) must be compared. Thus, write pointer has to be synchronized to receive clock and vice-versa. Thus, it is not data, but pointers that are synchronized. FIFO based synchronization is used when there is need for speed matching or data width matching. In case of speed matching, the faster port of FIFO normally handles burst transfers while slower part handles constant rate transfers. In FIFO based synchronization, average rate into and out of the FIFO are same in spite of different access speeds and types.


Signal indicating FIFO is full
Signal indicating FIFO is empty
Source domain Data bus
FIFO
Destination domain Data bus
Source domain Clock signal
Destination domain Clock signal

A FIFO based synchronization scheme

5. Why clock gating checks are needed? Explain clock gating checks for a multiplexer.

Solution:

Clock gating checks: Today's designs have many functional as well as test modes. A number of clocks propagate to different parts of design in different modes. And a number of control signals are there which control these clocks. These signals are behind switching on and off the design. Let us say, we have a simple design as shown in the figure below. Pin 'SEL' selects between two clocks. Also, 'EN' selects if clock will be propagating to the sub-design or not. Similarly, there are signals that decide what, when, where and how for propagation of clocks. Some of these controlling signals may be static while some of these might be dynamic. Even with all this, these signals should not play with waveform of the clock; i.e. these should not cause any glitch in clock path. There are both architectural as well as timing care-about that are to be taken care of while designing for signals toggling in clock paths. This scenario is widely known as 'clock gating'. The timing checks that need to be modeled in timing constraints are known as 'clock gating checks'.

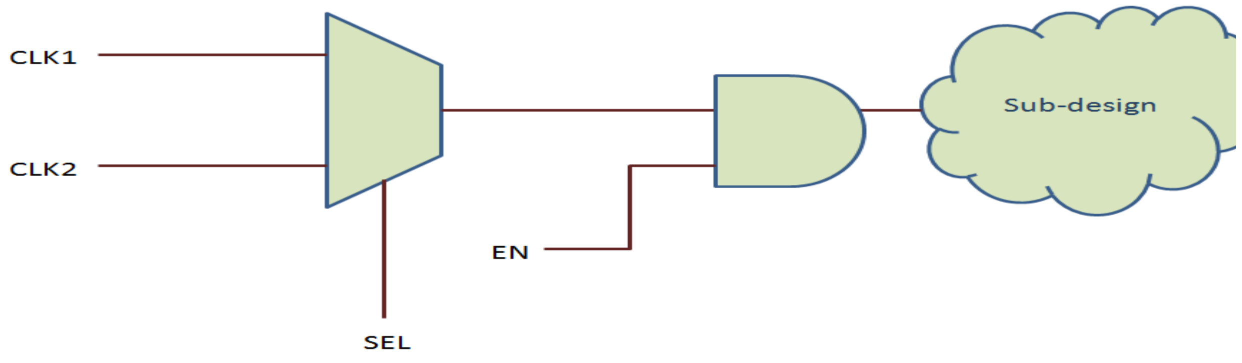


Figure 1: A simplest clocking structure

Definition of clock gating check: A clock gating check is a constraint, either applied or inferred automatically by tool, which ensures that the clock will propagate without any glitch through the gate.

Types of clock gating checks: Fundamentally, all clock gating checks can be categorized into two types:

AND type clock gating check: Let us say we have a 2-input AND gate in which one of the inputs has a clock and the other input has a data which will toggle while the clock is still on.

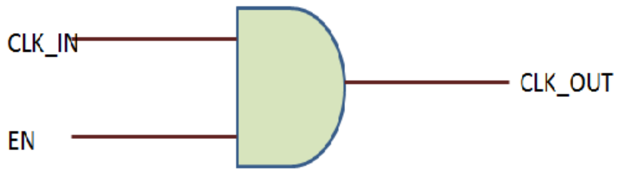


Figure 2: AND type clock gating check; EN signal controlling CLK_IN through AND gate

Since, the clock is free-running, we have to ensure that the change of state of enable signal does not cause the output of the AND gate to toggle. This is only possible if the enable input toggles when clock is at '0' state. As is shown in figure 3 below, if 'EN' toggles when 'CLK_IN' is high, the clock pulse gets clipped. In other words, we do not get full duty cycle of the clock. Thus, this is a functional architectural miss causing glitch in clock path. As is evident in figure 4, if 'EN' changes during 'CLK_IN' are low, there is no change in clock duty cycle. Hence, this is the right way to gate a clock signal with an enable signal; i.e. make the enable toggle only when clock is low.

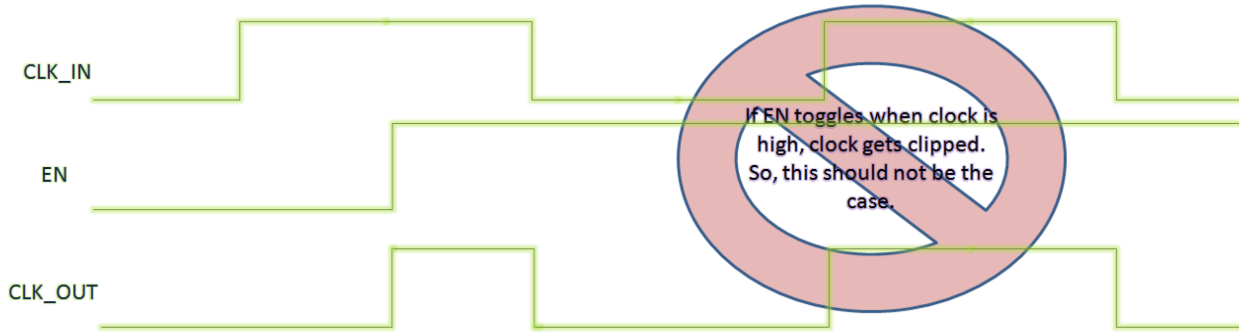

CLK_IN
EN
CLK_OUT
If EN toggles when clock is high, clock gets clipped. So, this should not be the case.

Figure 3: Clock being clipped when 'EN' changes when 'CLK_IN' is high

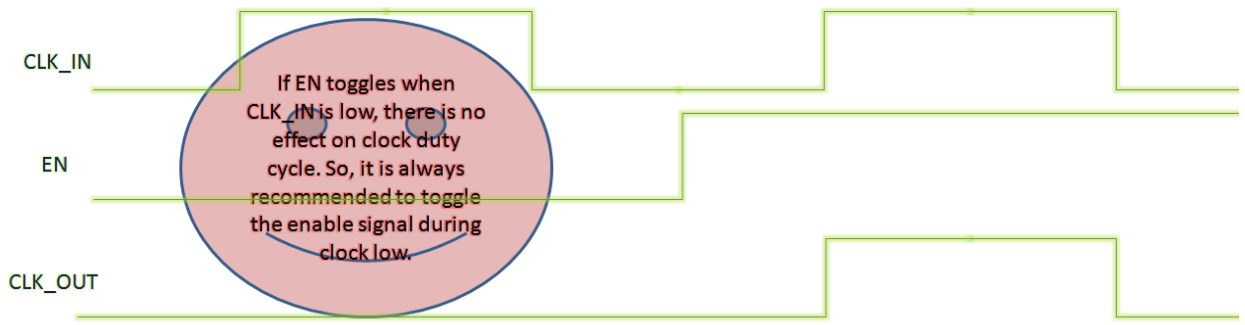

CLK_IN
EN
CLK_OUT
If EN toggles when CLK_IN is low, there is no effect on clock duty cycle. So, it is always recommended to toggle the enable signal during clock low.

Figure 4: Clock waveform not being altered when 'EN' changes when 'CLK_IN' is low

Theoretically, 'EN' can launch from either positive edge-triggered or negative edge-triggered flops. In case 'EN' is launched by a positive edge-triggered flop, the setup and hold checks will be as shown in figure 5. As shown, setup check in this case is on the next positive edge and hold check is on next negative edge. However, the ratio of maximum and minimum delays of cells in extreme operating conditions may be as high as 3. So, architecturally, this situation is not possible to guarantee the clock to pass under all conditions.

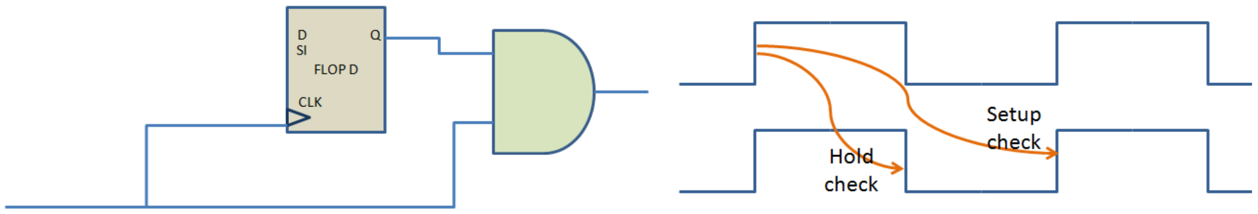


Figure 5: Clock gating setup and hold checks on AND gate when 'EN' launches from a positive edge-triggered flip-flop

On the contrary, if 'EN' launches from a negative edge-triggered flip-flop, setup check are formed with respect to the next rising edge and hold check is on the same falling edge (zero-cycle) as that of the launch edge. The same is shown in figure 6. Since, in this case, hold check is 0 cycle, both the checks are possible to be met for all operating conditions; hence, this solution will guarantee the clock to pass under all operating condition provided the setup check is met for worst case condition. The inactive clock state, as evident, in this case, is '0'.

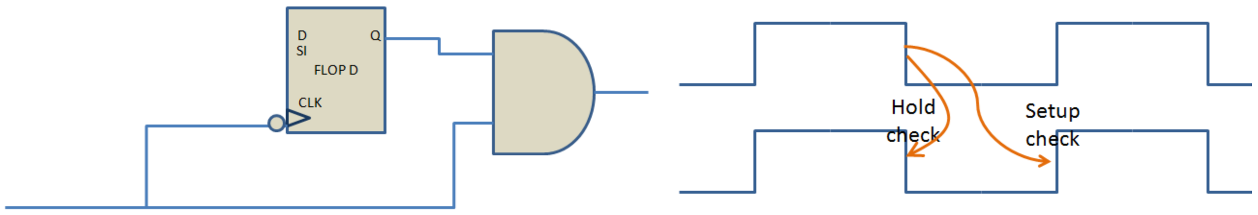


Figure 6: Clock gating setup and hold checks on AND gate when 'EN' launches from negative edge-triggered flip-flop

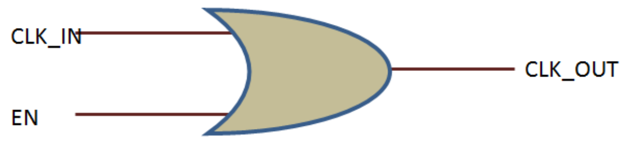


Figure 7: An OR gate controlling a clock signal 'CLK_IN'

OR type clock gating check: Similarly, since the off-state of OR gate is 1, the enable for an OR type clock gating check can change only when the clock is at '1' state. That is, we have to ensure that the change of state of enable signal does not cause the output of the OR gate to toggle. Figure 9 below shows if 'EN' toggles when 'CLK_IN' is high, there is no change in duty cycle. However, if 'EN' toggles when 'CLK_IN' is low (figure 8), the clock pulse gets clipped. Thus, 'EN' must be allowed to toggle only when 'CLK_IN' is high.

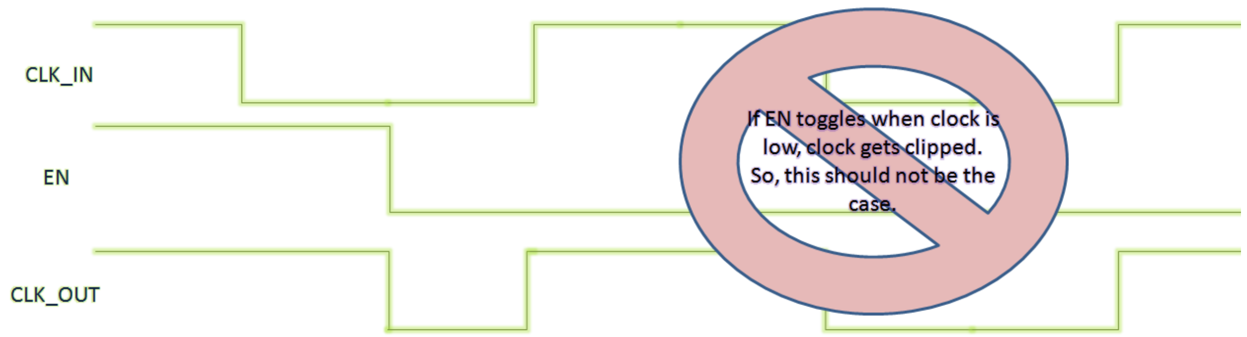

CLK_IN
EN
CLK_OUT
If EN toggles when clock is low, clock gets clipped. So, this should not be the case.

Figure 8: Clock being clipped when 'EN' changes when 'CLK_IN' is low

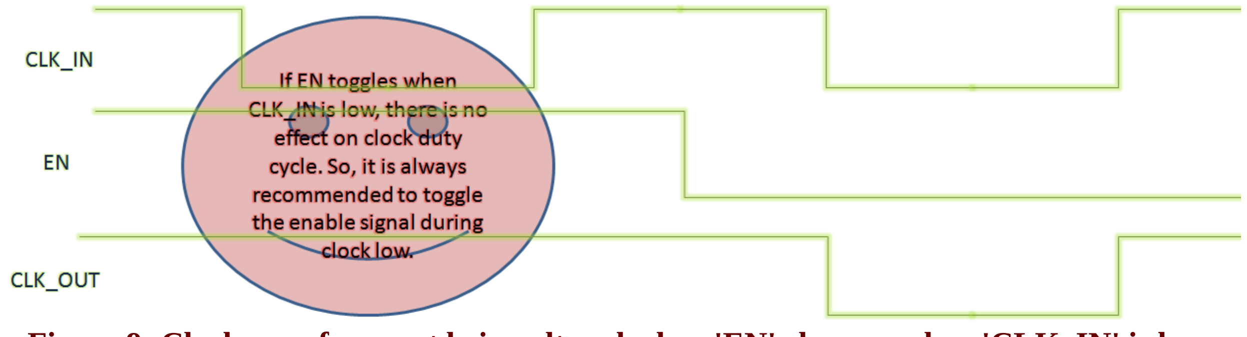

CLK_IN
EN
CLK_OUT
If EN toggles when CLK_IN is low, there is no effect on clock duty cycle. So, it is always recommended to toggle the enable signal during clock low.

Figure 9: Clock waveform not being altered when 'EN' changes when 'CLK_IN' is low

As in case of AND gate, here also, 'EN' can launch from either positive or negative edge flops. In case 'EN' launches from negative edge-triggered flop, the setup and hold checks will be as shown in the figure 10. The setup check is on the next negative edge and hold check is on the next positive edge. As discussed earlier, it cannot guarantee the glitch less propagation of clock.

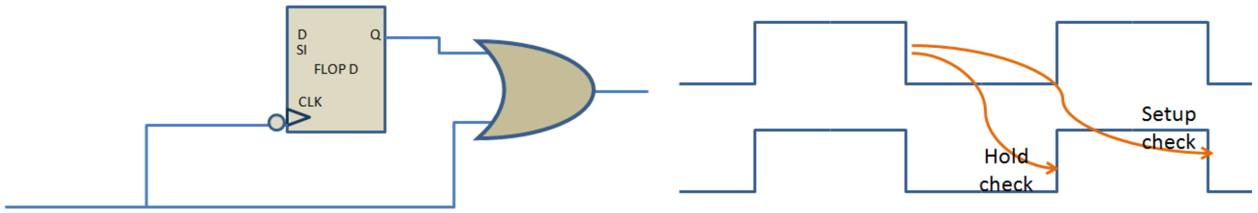


Figure 10: Clock gating setup and hold checks on OR gate when 'EN' launches from negative edge-triggered flip-flop

If 'EN' launches from a positive edge-triggered flip-flop, setup check is with respect to next falling edge and hold check is on the same rising edge as that of the launch edge. The same is shown in figure 11. Since, the hold check is 0 cycle, both setup and hold checks are guaranteed to be met under all operating conditions provided the path has been optimized to meet setup check for worst case condition. The inactive clock state, evidently, in this case, is '1'.

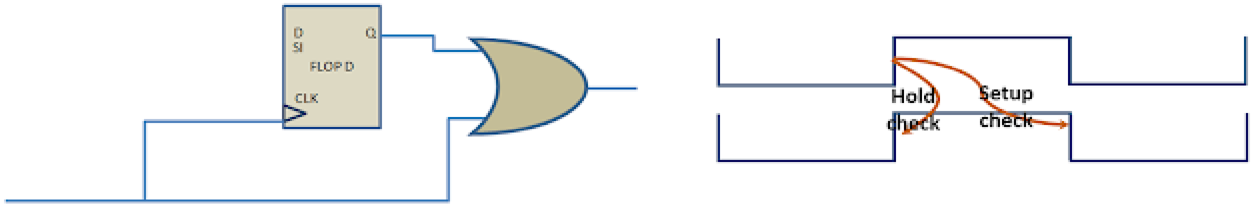


Figure 11: Clock gating setup and hold checks on OR gate when 'EN' launches from a positive edge-triggered flip-flop

We have, thus far, discussed two fundamental types of clock gating checks. There may be complex combinational cells other than 2-input AND or OR gates. However, for these cells, too, the checks we have to meet between the clock and enable pins will be of the above two types only. If the enable can change during low phase of the clock only, it is said to be AND type clock gating check and vice-versa.

The most common types of combinational cells with dynamic clock switching encountered in today's designs are multiplexers. We will be discussing the clock gating checks at a multiplexer. For simplicity, let us say, we have a 2-input multiplexer with 1 select pin. There can be two cases:

Case 1: Data signal at the select pin of MUX used to select between two clocks

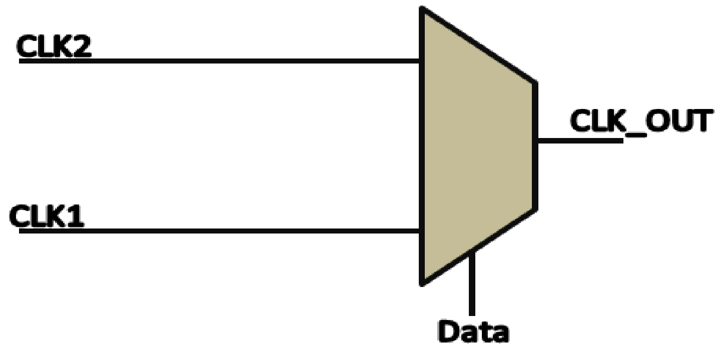


Figure 1: MUX with Data as select dynamically selecting the clock signal to propagate to output

This scenario is shown in figure 1 above. This situation normally arises when ‘Data’ acts as clock select and dynamically selects which of the two clocks will propagate to the output. The function of the MUX is given as:

CLK_OUT = Data.CLK1 + Data’.CLK2

The internal structure (in terms of basic gates) is as shown below in figure 2.

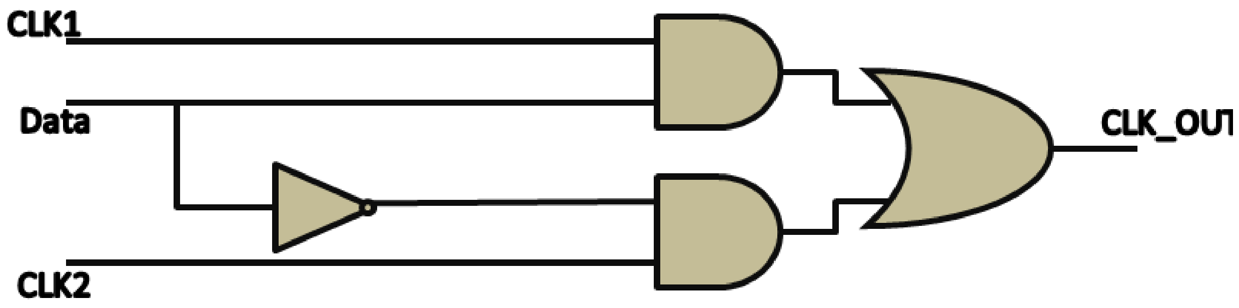


Figure 2: Internal structure of mux in figure 1

There will be two clock gating checks formed:

1. **<u>Between CLK1 and Data</u>:** There are two cases to be considered for this scenario:

    - **When CLK2 is at state '0'**: In this scenario, if the data toggles when CLK1 is '0', it will pass without any glitches. On the other hand, there will be a glitch if data toggles when CLK1 is '1'. Thus, the mux acts as AND gate and there will be AND-type clock gating check.
    - **When CLK2 is '1'**: In this scenario, if data toggles when CLK1 is '1', it will pass without any glitches; and will produce a glitch if toggled when CLK1 is '0'. In other words, MUX acts as an OR gate; hence, OR-type clock gating check will be formed in this case.

2. Between CLK2 and Data: This scenario also follows scenario '1'. And the type of clock gating check formed will be determined by the state of inactive clock.

Thus, the type of clock gating check to be applied, in this case, depends upon the inactive state of the other clock. If it is '0', AND-type check will be formed. On the other hand, if it is '1', OR-type check will be formed.

Case 2: Clock signal is at select line. This situation is most common in case of Mux-based configurable clock dividers wherein output clock waveform is a function of the two data values.

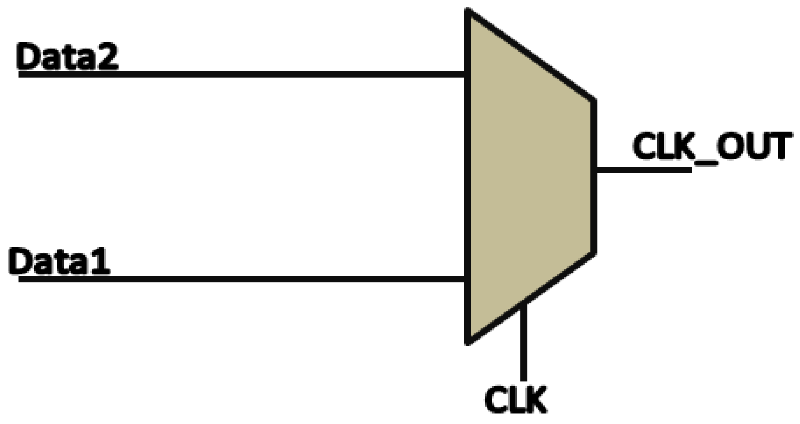


Figure 3: Combination of Data1 and Data2 determines if CLK or CLK' will propagate to the output

In this case too, there will be two kinds of clock gating checks formed:

i) **Between CLK and Data1**: Here, both CLK and Data1 are input to a 2-input AND gate, hence, there will be **AND type check between CLK and Data1**. The following SDC command will serve the purpose:

set_clock_gating_check -high 0.1 [get_pins MUX/Data1]

The above command will constrain an AND-type clock gating check of 100 ps on Data1 pin.

ii) **Between CLK and Data2**: As is evident from figure 3, there will be AND type check between CLK' and Data2. This means Data2 can change only when CLK' is low. In other words, Data2 can change only when CLK is high. This means there is **OR type check between CLK and Data2.** The following command will do the job:

set_clock_gating_check -low 0.1 [get_pins MUX/Data2]

The above command will constrain an OR-type clock gating check of 100 ps on Data2 pin.

Thus, we have discussed how there are clock gating checks formed between different signals of a MUX.