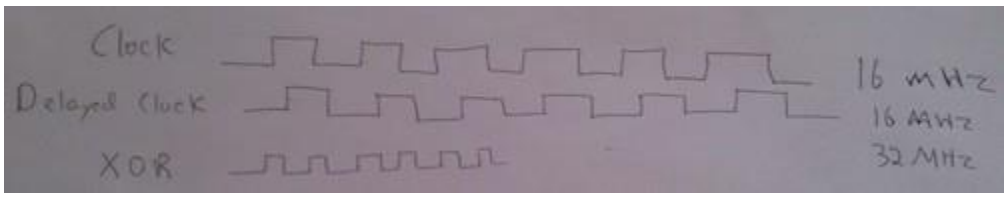
Assignment 5 Solutions

1. Make a simple circuit whose output clock is four times higher in frequency to the input clock

Solution:

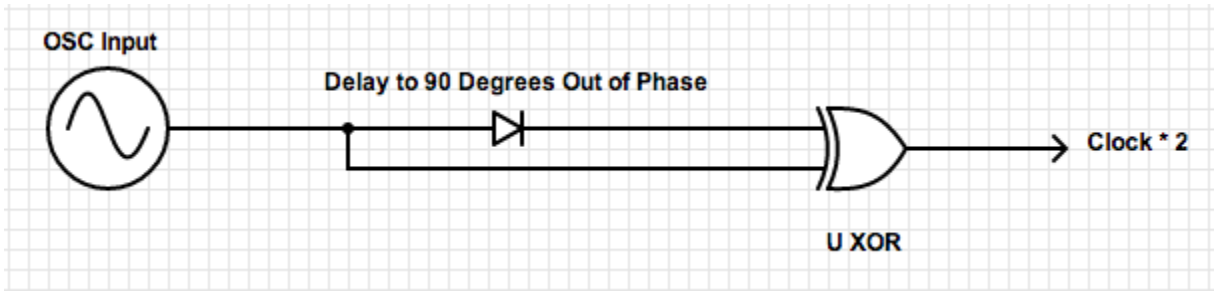
For the uninitiated, ***XOR***, or ***exclusive or***, is a function which will return 'true' when and odd number of inputs are 'true'.

For a two input XOR function, that means **only one will be high**. Don't worry, I drew it for you:

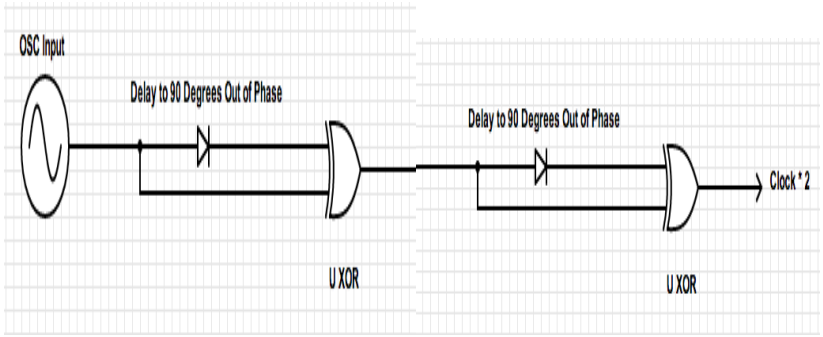


So, the clock will be high when only one of the two out of phase clocks is high… which happens to be twice as often as the original clock.

And that's the theory, which again I've conveniently drawn for you here:

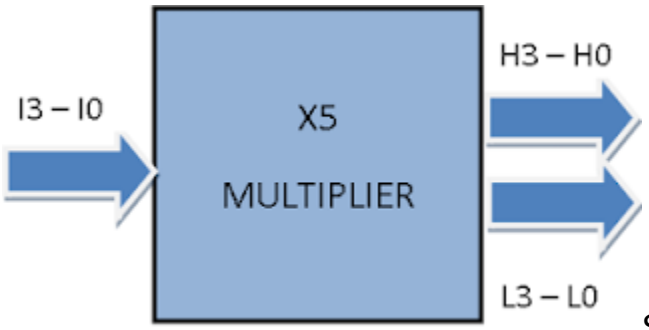


This circuit provides the clock multiple by 2. You add another of this circuit in series to get the multiply by 4 clock circuit.



2. **Make one digital circuit which takes a BCD digit as input and produces the output that is equal to 3 times of the input digit in the same format.**

Solution: Solution – Binary Coded Decimal (BCD) is the format to represent decimal numbers in binary form. In this format, each digit of the decimal number is coded separately into 4-bit binary number. So, only binary values from 0000 to 1001 are valid codes. E.g. A two digit decimal number will be coded into BCD code with 8 binary digits.



Since, the circuit produces output that is 5 times the BCD digit, so, output value can contain maximum of 2 decimal digits OR 8 binary digits. Let I3-I0 be the 4-bit BCD digit input to the X5 multiplies and H3-H0 and L3-L0 be the 8-bit BCD output as shown in the figure below.

The truth table for X5 multiplier can be represented as shown in the figure below:

BCD input	BCD output	I3	I2	I1	I0	H3	H2	H1	H0	L3	L2	L1	L0
0	00	0	0	0	0	0	0	0	0	0	0	0	0
1	05	0	0	0	1	0	0	0	0	0	1	0	1
2	10	0	0	1	0	0	0	0	1	0	0	0	0
3	15	0	0	1	1	0	0	0	1	0	1	0	1
4	20	0	1	0	0	0	0	1	0	0	0	0	0
5	25	0	1	0	1	0	0	1	0	0	1	0	1
6	30	0	1	1	0	0	0	1	1	0	0	0	0
7	35	0	1	1	1	0	0	1	1	0	1	0	1
8	40	1	0	0	0	0	1	0	0	0	0	0	0
9	45	1	0	0	1	0	1	0	0	0	1	0	1
X	XX	1	0	1	0	X	X	X	X	X	X	X	X
X	XX	1	0	1	1	X	X	X	X	X	X	X	X
X	XX	1	1	0	0	X	X	X	X	X	X	X	X

X	XX	1	1	0	1	X	X	X	X	X	X	X	X
X	XX	1	1	1	0	X	X	X	X	X	X	X	X
X	XX	1	1	1	1	X	X	X	X	X	X	X	X

Drawing K-maps for all the outputs:

K Map for H3, L3 and L1

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

=> H3 = L3= L1 = 0

Similarly, by solving using K-Maps for all the outputs, we get

H2 = I3

H1 = I2

H0 = I1

L2 = L0 = I0

Thus, all the outputs are either a direct connection from one of the inputs or zero. So, the whole function can be implemented without using a single logic gate. Sounds strange!!!

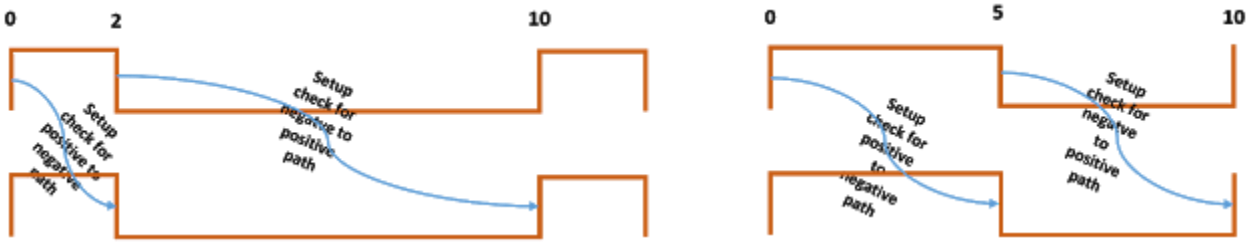
Can you come up with a better solution for this problem? Let us know with your comments.

3. What is duty cycle? Explain in details how duty cycle impacts on timing analysis of any circuit design.

Duty cycle: Duty cycle of a clock is defined as the fraction of a period of clock during which the clock is in active state. Duty cycle of a clock is normally expressed as a percentage. For instance, figure below shows a clock having an active state of '1' stays low for 2 ns during its period of 10 ns. It is, therefore, said to have a duty cycle of 20%.

How duty cycle impacts timing: Duty cycle of clock plays a big role in timing closure of designs. We need to consider following factors related to duty cycle variation while timing:

- **Half cycle timing paths**: If there are both positive and negative edge-triggered flip-flops in the design, duty cycle of the clock matters a lot. For instance, if we have a clock of 100 MHz with 20% duty cycle; For a timing path from positive edge-triggered flip-flop to negative edge-triggered flip-flop, we get only 2 ns for setup timing for positive-to-negative path and 8 ns for negative-to-positive path as compared to 10 ns for a full cycle path. However, if the same clock had duty cycle of 50%, we would have got 5 ns for the same half cycle timng path.



- **Minimum pulse width requirements**: At high frequencies, duty cycle matters a lot. For instance, every sequential element has requirement of minimum pulse width that should reach it (read this). If the duty cycle of the clock is not close to 50%, we are limited in providing high frequency even if we are capable of meeting timing at even higher frequencies. Let us take an example. If the minimum pulse width requirement of a flip-flop is 500 ps, then with 50% duty cycle clock, we can use a clock of 1 GHz (1 ns clock period). But if we use a clock of duty cycle of 20%, we cannot use a clock greater than 400 MHz.

With the above things in mind, it makes sense to use a clock with duty cycle as close to 50%. However, in many scenarios, it may not be feasible to do so. So, one needs to decide the priorities; i.e., architecture complexities vs timing complexities. Generating a divided clock of 50% duty cycle is not always possible and there are a few complexities involved in architecture. For instance, clock waveform synchronization between the clocks if there are multiple dividers. Also, for odd division factors like divide_by_3 etc., we need more complex divider circuitry than what may be required for divide_by_2 or divide_by_4 etc.

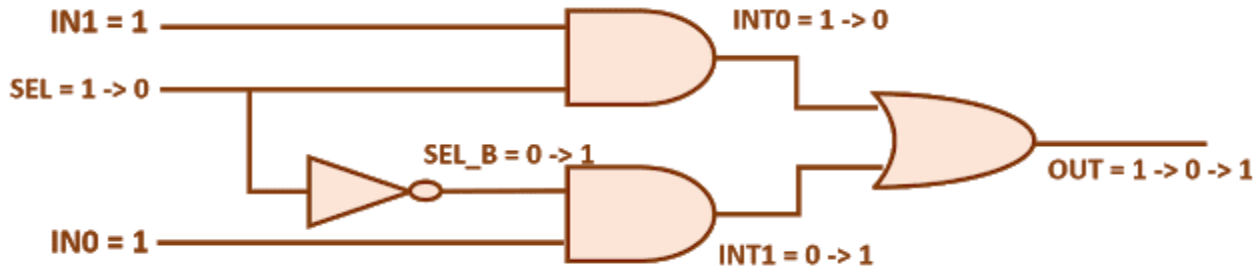
4. **How can you implement one and gate using multiplexer? A 2-input multiplexer has both of its input getting value of 1. Will there be any toggle (glitch) happening at the output of the mux? If yes is it expected? What if both the inputs are getting value 0.**

   Solution:

We all know that a multiplexer's output is equal to
IN0 if SEL = 0

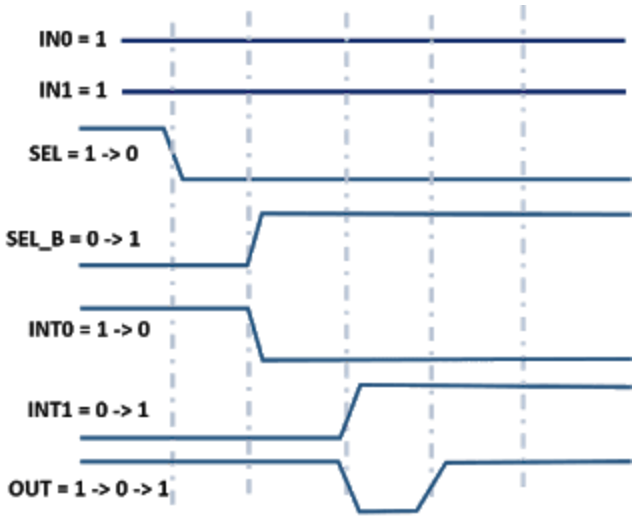
IN1 if SEL =1

So, if both IN0 and IN1 are getting same logic value, output must not toggle. However, if we observe carefully, there is a high chance of a momentary glitch at the output in case both inputs are at value "1" and select toggles from "1" to "0". To understand this, we need to look into the internal structure of the mux, which is as shown in figure 2 below.



The figure says that output goes momentarily to "0" before finally settling down to "1". Why is this so? The reason behind this is the two paths going from SEL to OUT and toggling of both the inputs of final OR gate. And there is asymmetry of delays with one inverter being extra in one of the paths. This causes the output of the mux to go momentarily to zero.

Let us analyze this with the help of timing waveforms (assuming delay of each element to be 1 unit):



Thus, it is clear from the timing waveforms that there is a glitch in the output. It is possible to minimize the extent of this glitch by minimizing the difference of delays between the two paths getting formed between the SEL and OUT. However, it cannot be guaranteed even with greatest of precision during design as there are mismatches in fabrication of individual gates. So, even the best of multiplexers will have this limitation, however small it may be, unless designed specifically for this purpose. Can you suggest a design improvement that can help in this scenario?

One is forced to think here that what can be the consequences of such a glitch and what remedies can be there for this. I had written a post Glitches in combinational circuits that discussed what can be the consequences of glitches in combinational circuits. This scenario is a special case, but with some twist. Let us discuss all the cases one by one.

- If this case is in a data path for synchronous circuits, there is no issue as discussed in one of the points in our post Glitches in combinational circuits.
- If this case happens for a data path in asynchronous circuits, this can be an issue. So, synchronization circuits have to be designed with utmost care and following the rules of data synchronization
- If this scenario occurs in either path of clock or reset, this is an issue as this glitch can alter the state of the design by either letting the flop capture data at "D" pin by acting as an extra clock pulse, or can reset the flop.

5. What is setup time and hold time? What is the cause of origin of setup and hold time? How do you check setup and hold time in a design? How to mitigate the setup and hold time violations?

Solution:

In digital designs, each and every flip-flop has some restrictions related to the data with respect to the clock in the form of windows in which data can change or not. There is always a region around the clock edge in which input data should not change at the input of the flip-flop. This is because, if the data changes within this window, we cannot guarantee the output. The output can be the result of either of the previous input, the new input or metastability (as explained in our post 'metastability'). This window is marked by two boundary lines, one pertaining to the setup time of the flop, the other to the hold time defined as below.

Definition of Setup time: Setup time is defined as the minimum amount of time before the clock's active edge that the data must be stable for it to be latched correctly. In other words, each flip-flop (or any sequential element, in general) needs some time for the data to remain stable before the clock edge arrives, such that it can reliably capture the data. This duration is known as **setup time**.

The data that was launched at the previous clock edge should be stable at the input at least setup time before the clock edge. So, adherence to setup time ensures that the data launched at previous edge is captured properly at the current edge. In other words, we can also say that setup time adherence ensures that the system moves to next state smoothly.

Definition of Hold time: Hold time is defined as the minimum amount of time after the clock's active edge during which data must be stable. Similar to setup time, each sequential element needs some time for data to remain stable after clock edge arrives to reliably capture data. This duration is known as **hold time**.

The data that was launched at the current edge should not travel to the capturing flop before hold time has passed after the clock edge. Adherence to hold time ensures that the data launched at current clock edge does not get captured at the same edge. In other words, hold time adherence ensures that system does not deviate from the current state and go into an invalid state.

As shown in the figure 1 below, the data at the input of flip-flop can change anywhere except within the seup time hold time window.

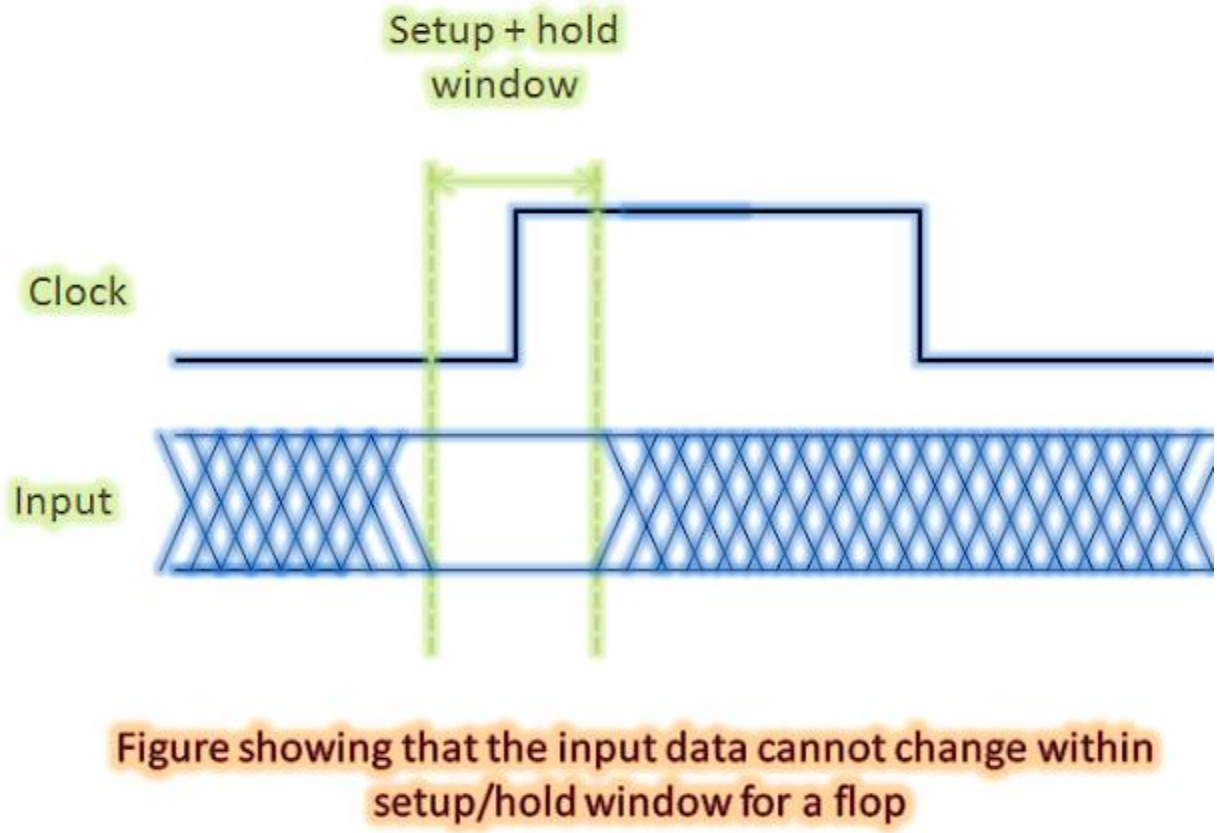
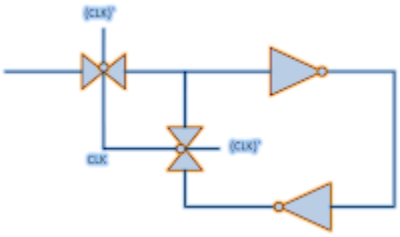


Figure 1: Setup-hold window

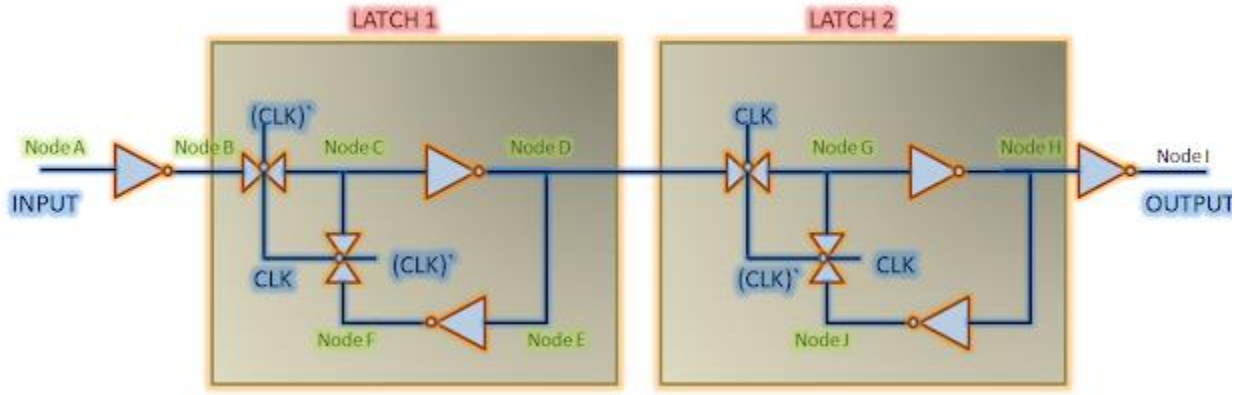


A D-type latch

Cause/origin of setup time and hold time: Setup time and hold time are said to be the backbone of timing analysis. Rightly so, for the chip to function properly, setup and hold timing constraints need to be met properly for each and every flip-flop in the design. If even a single flop exists that does not meet setup and hold requirements for timing paths starting from/ending at it, the design will fail and meta-stability will occur. It is very important to understand the origin of setup time and hold time as whole design functionality is ensured by these. Let us discuss the origin of setup

time and hold time taking an example of D-flip-flop as in VLSI designs, D-type flip-flops are almost always used. A D-type flip-flop is realized using two D-type latches; one of them is positive level-sensitive, the other is negative level-sensitive. A D-type latch, in turn, is realized using transmission gates and inverters. Figure below shows a positive-level sensitive D-type latch. Just inverting the transmission gates' clock, we get negative-level sensitive D-type latch.

A complete D flip-flop using the above structure of D-type latch is shown in figure below:

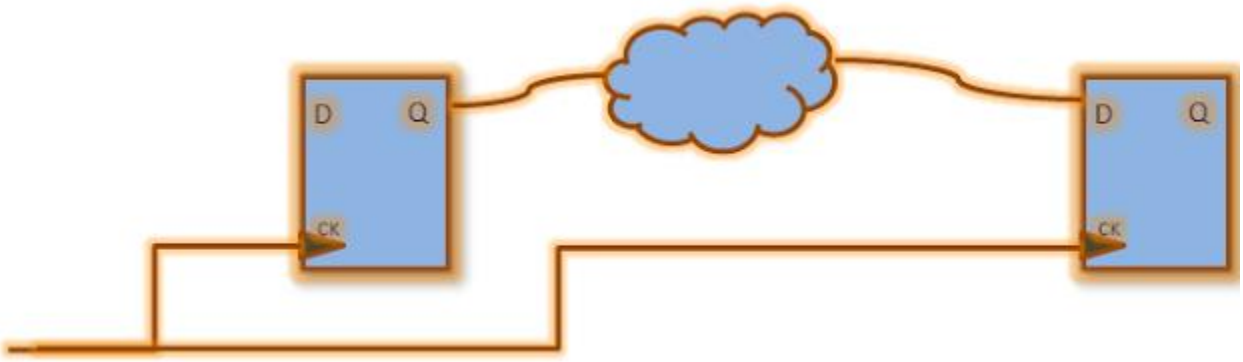


A D-type flip-flop

Now, let us get into the details of above figure. For data to be latched by 'latch 1' at the falling edge of the clock, it must be present at 'Node F' at that time. Since, data has to travel **'NodeA' -> 'Node B' -> 'Node C' -> 'Node D' -> 'Node E' -> 'Node F'** to reach 'Node F', it should arrive at flip-flop's input (Node A) at some earlier time. This time for data to reach from 'Node A' to 'Node F' is termed as data setup time (assuming CLK and CLK' are present instantaneously. If that is not the case, it will be accounted for accordingly). Similarly, it is necessary to ensure a stable value at the input to ensure a stable value at '**Node C**'. In other words, hold time can be termed as delay taken by data from **'Node A' to 'Node C'**.

Setup and hold checks in a design: Basically, setup and hold timing checks ensure that a data launched from one flop is captured at another properly. Considering the way digital designs of

today are designed (finite state machines), the next state is derived from its previous state. So, data launched at one edge should be captured at next active clock edge. Also, the data launched from one flop should not be captured at next flop at the same edge. These conditions are ensured by setup and hold checks. Setup check ensures that the data is stable before the setup requirement of next active clock edge at the next flop so that next state is reached. Similarly, hold check ensures that data is stable until the hold requirement for the next flop for same clock edge has been met so that present state is not corrupted.



A sample path in a design

Shown above is a flop-to-flop timing path. For simplicity, we have assumed that both the flops are rise edge triggered. The setup and hold timing relations for the data at input of second flop can be explained using the waveforms below:

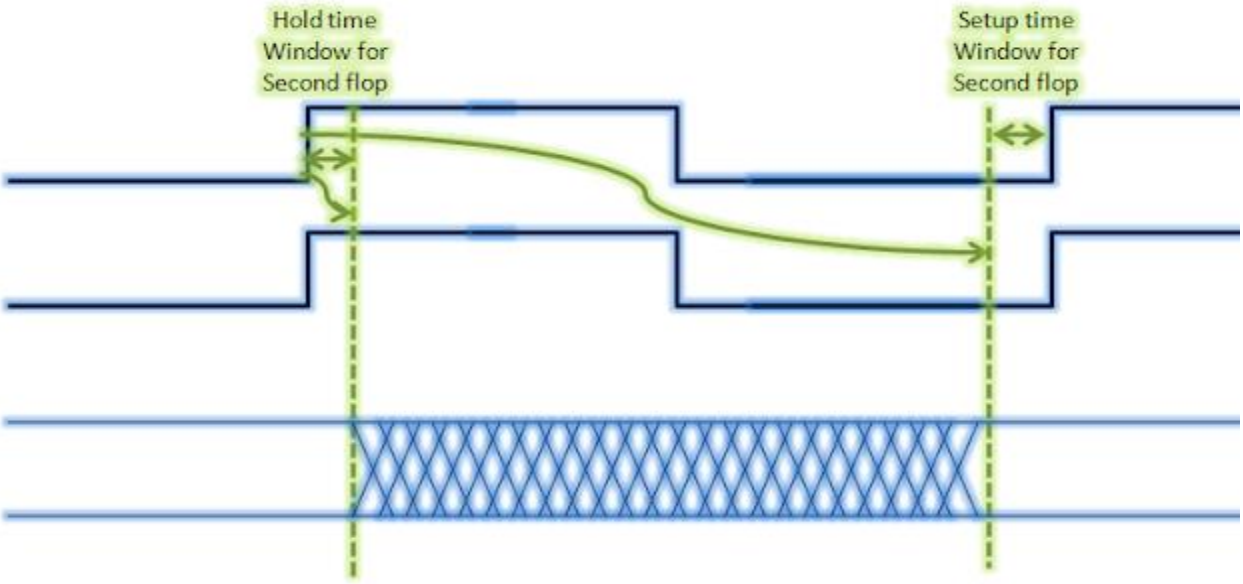


Figure showing setup and hold checks being applied for the timing path shown above

As shown, data launched from launching flop is allowed to arrive at the input of the second flop only after a delay greater than its hold requirement so that it is properly captured. Similarly, it must not have a delay greater than (clock period – setup requirement of second flop). In other words, mathematically speaking, setup check equation is given as below (assuming zero skew between launch and capture clocks):

$$T_{ck->q} + T_{prop} + T_{setup} < T_{period}$$

Similarly, hold check equation is given as:

$$T_{ck->q} + T_{prop} > T_{hold}$$

If we take into account skews between the two clocks, the above equations are modified accordingly. If T_{skew} is the skew between launch and capture flops, (equal to latency of clock at capture flop minus latency of clock at launch flop so that skew is positive if capture flop has larger latency and vice-versa), above equations are modified as below:

$$T_{ck->q} + T_{prop} + T_{setup} - Tskew < T_{period}$$

$$T_{ck->q} + T_{prop} > T_{hold} + T_{skew}$$

Setup checks and hold checks for reg-to-reg paths explains different cases covering setup and hold checks for flop-to-flop paths.

What if setup and/or hold violations occur in a design: As said earlier, setup and hold timings are to be met in order to ensure that data launched from one flop is captured properly at the next flop at next clock edge so as to transfer the state-machine of the design to the next state. If the setup check is violated, the data will not be captured at the next clock edge properly. Similarly, if hold check is violated, data intended to be captured at the next edge will get captured at the same edge. Setup hold violations can also lead to data changing within setup/hold window of the capturing flip-flop. It may lead to metastability failure in the design (as explained in our post 'metastability'). So, it is necessary to have setup and hold requirements met for all the flip-flops in the design and there should not be any setup/hold violation.

What if you fabricate a design without taking care of setup/hold violations: If you fabricate a design having setup violations, you can still use it by lowering the frequency as the equation involves the variable clock frequency. On the other hand, a design with hold violation cannot be run properly. So, if you fabricate a design with an accidental hold violation, you will have to simply throw away the chip (unless the hold path is half cycle as explained here). A design with half cycle hold violations only can still be used at lower frequencies.

Tackling setup time violation: As given above, the equation for setup timing check is given as:

$$T_{ck->q} + T_{prop} + T_{setup} - T_{skew} < T_{period}$$

The parameter that represents if there is a setup time violation is **setup slack**. The setup slack can be defined as the difference between the L.H.S and R.H.S. In other words, it is the margin that is available such that the timing path meets setup check. The setup slack equation can be given as:

$$\text{Setup slack} = T_{period} - (T_{ck->q} + T_{prop} + T_{setup} - T_{skew})$$

If setup slack is positive, it means there is still some margin available in the timing path. On the other hand, a negative slack means that the paths violates setup timing check by the amount of setup slack. To get the path met, either data delay should be decreased or clock period should be increased.

Mitigating setup violation: Thus, we can meet the setup requirement, if violating, by

1. Decreasing clk->q delay of launching flop
2. Decreasing the propagation delay of the combinational cloud
3. Reducing the setup time requirement of capturing flop
4. Increasing the skew between capture and launch clocks
5. Increasing the clock period

Tackling hold time violation: Similarly, the equation for hold timing check is as below:

$$T_{ck->q} + T_{prop} > T_{hold} + T_{skew}$$

The parameter that represents if there is a hold timing violation is **hold slack**. The hold slack is defined as the amount by which L.H.S is greater than R.H.S. In other words, it is the margin by which timing path meets the hold timing check. The equation for hold slack is given as:

Hold slack $= T_{ck->q} + T_{prop} - T_{hold} - T_{skew}$

If hold slack is positive, it means there is still some margin available before it will start violating for hold. A negative hold slack means the path is violating hold timing check by the amount represented by hold slack. To get the path met, either data path delay should be increased, or clock skew/hold requirement of capturing flop should be decreased.

Mitigating hold violation: We can meet the hold requirement by:

1. Increasing the clk->q delay of launching flop
2. Decreasing the hold requirement of capturing flop
3. Decreasing clock skew between capturing clock and launching flip-flops