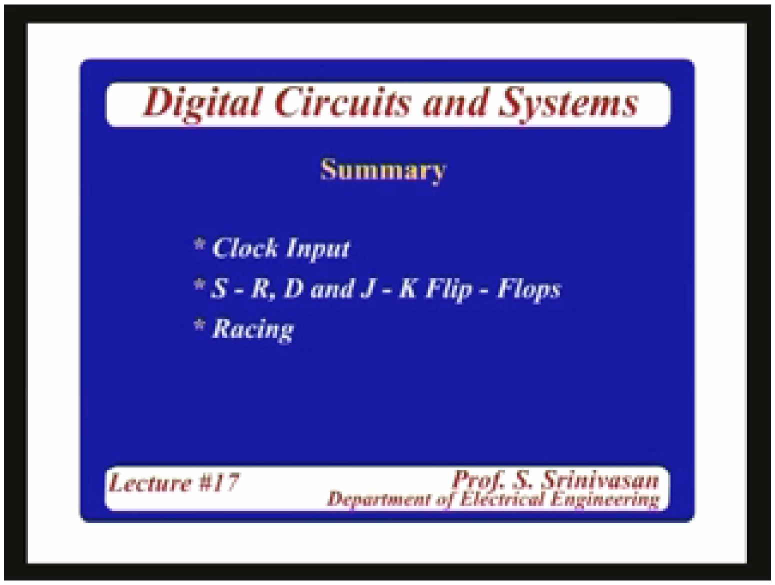
Digital Circuits and Systems
Prof. S. Srinivasan
Electronics and Communication Engineering
Indian Institute of Technology Madras
S-R, J-K and D Flip Flops
Lecture # 17

(Refer Slide Time: 1:27)



Digital Circuits and Systems

Summary

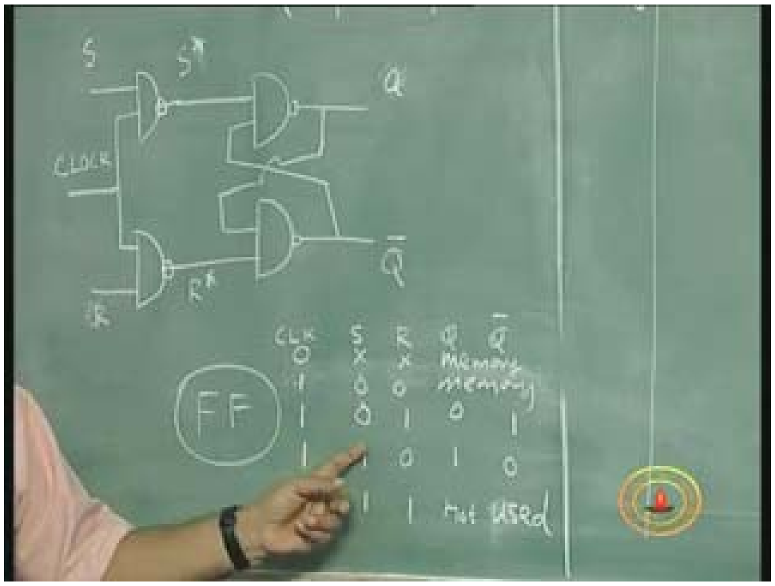
- * Clock Input
- * S - R, D and J - K Flip - Flops
- * Racing

Lecture #17 Prof. S. Srinivasan
Department of Electrical Engineering

In the last lecture we introduced the concept of storage or the memory element which is needed for sequential circuits because in a sequential circuit the output is determined not only with the presents of the inputs but also by the past outputs of the circuit. Naturally we need storage elements in order to remember the previous outputs and feed it as additional inputs to the system. And the circuit we saw was a simple latch where 1 bit can be stored that’s all we need any way. In digital systems we only use bits. As long as you have a circuit to store a bit of information for 0 or 1 you can replicate it and use it as many times as you want.

Today we will see some improvements on that basic latch it was called. First thing you have to make is to introduce what is known as a clock. So, clocked latch is also called a flip-flop. When there is a set of inputs you applied to store to the circuit 0 or 1 because we said that for the RS flip-flop or SR flip-flop you have to give 0 1 combination in order to store 1 or 0 so you need to apply the pair of inputs in order to store the bit that you want, and when it changes the output changes.

(Refer Slide Time 20:50)



CLK	S	R	Q	$\overline{Q}$
0	X	X	Memory	
1	0	0	Memory	
1	0	1	0	1
1	1	0	1	0
[illegible]	1	1	not used	

When a circuit is working in the overall system you can't afford to change the inputs very frequently. The system has to settle down. Once you store that information that will be recalled only when it's needed by the subsequent circuitry to the circuit which uses this and if in the mean time the input changes what you have originally put would have changed. So, in other words you want a control of determining when to store the information if you leave the circuit as it is any changes in the input will naturally change the output, sometimes it may be intentional sometimes it may not be intentional so we want to have a control of this is the information I have stored so I will keep it for a while and when I need it I will use it then I will go and change the input to another value which will be used in a subsequent transaction. That means I need to control the storage mechanism this control is called the clock.

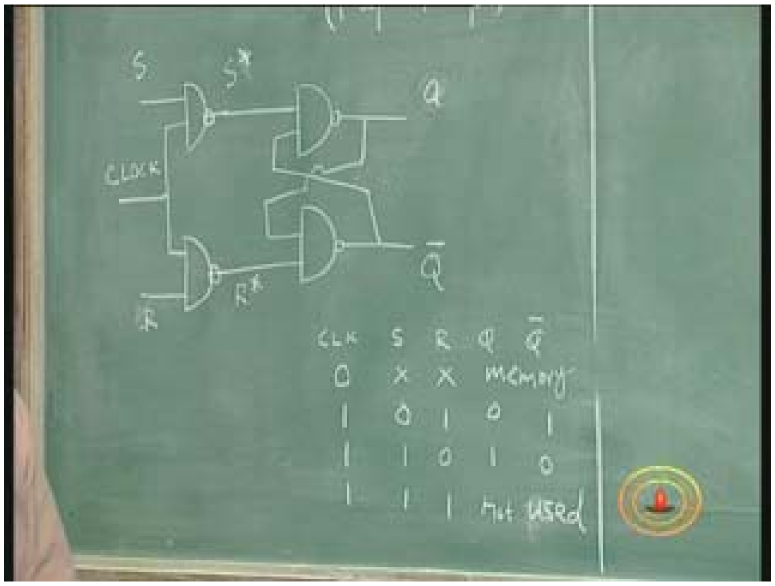
That means it is not enough if you apply the inputs. The inputs will be recognized and saved or stored in the latch as 1 or 0 only if the inputs are present and you also apply your enabling input or a gating input or a clock input. Therefore in other words what I am saying is if I take that simple NAND latch, we saw two circuits in the last lecture and one of them is this (Refer Slide Time: 6:21) where this is a set input and there is a reset input.

Of course in this case if this set is 0 and R is 1 and set is 0 and reset is 1 Q is 1 and when set is 1 and reset is 0 Q is 0, we didn't want it. Actually the set reset should have been, when set is 1 and R is 0 output should be 0 I mean when set is 1 and reset is 0 we want the output to be 1, and when set is 0 and reset is 1 we want the output to be reset to 0. I could have still used NOR gate combination or I could have re-designated this as S bar R bar whatever as I said yesterday in the last lecture. but that is not the point here, the point here is I do not want any change in the S and R to affect this storage mechanism, there is a bit stored here there is 0 and 1 and when you remove the input the bit remains stored but I want that to be effective by a control mechanism and I want to incorporate that. That control mechanism is a gating signal and only when the gating signal is enabled or gating signal is high then whatever be the S and R combination the output will change accordingly. If the gating mechanism signal is not enabled then whatever values you give to the inputs does not affect what is already stored in it but it will remain as a memory state. So a very simple way of doing this is to put AND gate to each of this and instead of giving S and R here I can give S and R here (Refer Slide Time: 8:11) at these two places and if you want to call this S star R star the gated output and the second input to this will come from a signal which is called a clock signal.

If clock is high SR gets reflected here, the S star is same as S and R star is same as R if clock is 0 S star and R star are both 0 but we do not want that, instead we want this to work only with the gating possibility and you can also remove that combination of that S bar R bar I thing by putting instead of an AND gate a NAND gate. If I put a NAND gate here there is a natural inversion because of the NAND gate so now it becomes a true SR latch. That means now the truth table of this would be clock S R Q Q bar, and if clock is 0 disabled whatever be the value of S and R because it is going to remain under 1 1 state in a NAND gate you put 0 one of them is 0 and the output is 1 so if it is a 1 1 state then it is a memory state. For this latch 1 1 state it is the memory state so this becomes a memory state.

That means your S and R inputs the set and reset inputs will not affect these storage, the latch. The latch continues to store what it had earlier. If you want to put a new value and store it first thing I have to do is to enable that clock signal by making clock as equal to 1 then in this case S0 R1 becomes 0 and is not permitted or not used.

(Refer Slide Time: 10:55)



CLK	S	R	Q	$\bar{Q}$
0	X	X	memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	not used	

This is SR latch I can even put 0 0 I forgot to add this may be I can add one more row here it is a bit crowded but what I mean is both are 0 memory state. This is the normal operation of SR latch, 0 0 is memory state and 0 1 is a reset state 1 0 is a set state, 1 1 is a not used state and all if clock is 1 all if clock signal or enable signal is 1. If on the other hand the clock signal is 0 or absent then whatever value you try to put in the S and R is not going to affect the outputs and the output is going to remain as it was earlier which is called memory state. So this clocking is a gating signal or enabling signal which lets you decide when to put a new value of the input and store it in the latch. After putting it and you don't want the changed value at all for a while you want to keep it for a while and then use it later on all you have to do is to disable this gate or make this clock 0 and then it will continue to be stored as it was and any changes in the input accidentally happening or intentionally will not affect the storage.

Why do we need this clocking operation? As I said first the output of the latch may be the input of a next circuit subsequent circuit and any frequent change intentional or unintentional will not go through a chain reaction and go to the next state. Normally in a sequential circuit we have several things happening and there must be some control or some timing or clocking mechanism as they call it a timing precisely at which point in time everything will change, there may be several stages of this circuit and each of this stage will have to take the inputs from the previous stage and process it and give an output to the next stage but they cannot work at random.

You work at random some of them may work faster than the other and some of them may take a previous output from the previous stage and process it even before the present output is available. But if you try to synchronize the operation of all the subsystems or circuits within the sequential system then you have control over how to transmit data through a chain of these circuits or subsystems. So, in other words you sort of sequence the operations the way you want.

Since this is a sequential circuit you want some sort of a control, clocking, timing etc the timing has to be controlled by you. You may want to do it faster I can do it very very fast, nobody said I should not do it very fast, the clocking can be removed and put very very fast. If I want to work at a very high speed I can give a very high clock signal. If you want to work with very low speed I can give a clocking signal which is of a very low frequency so that is possible. That is why it is called a clock because it sort of times the whole thing, it precisely controls like our clocks, we know when to do what.

If the class starts at 4 I will set the clock to 4 and the class will start then so that people come earlier, of course people come late, at least people come earlier. I come, I start the class and then the people keep walking in, so some sort of regularity is put into the behavior which helps in the overall system performance, it determines system performance.

What is a clock?

Clock is nothing but a signal which goes from high to low. All I want is the clock to be 0. So clock signal is nothing but a signal if you write in terms of the time if you draw the clock signal as a functional time it's a signal which goes from 0 to 1 to 0 to 1 and whatever is the value of 0 and 1 defined in your hardware as I said one general case is 0V and 5V or 0V and 3.3V so I will simply put 0 and 1 I will not venture to write the values because it depends on the logic levels you are working with, 0 state and one state where 0 is the clock off and 1 is the clock on it is the clock low and clock high. So the latch will work only when the clock is high. So the best behavior is of the latch can be achieved if you make the inputs available before the clock becomes 1. The clock is still 0 and you make the inputs then let the clock take the high value so that the input will be transmitted into the latch the proper value would be stored and do not make any change in the input during the period of the clock active high. When the clock is high if you make changes input that will reflect at the output. I do not want that I do not want more than one change in one clock period, I don't want more than one change in the output because the next circuit does not know how frequently should it expect a change.

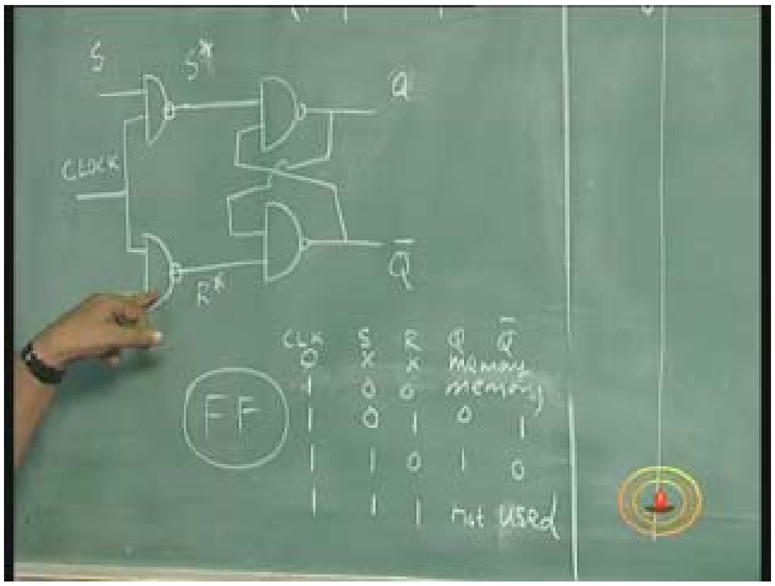
I am feeding this output to the next circuit and the next circuit assumes that the 1s in a clock cycle it is going to change so I will take that into account and process it further. And as I was doing it and processing the previous value and suddenly the value changes again there is no reliability, I always say reliability is more important. So if you do not know how it is going to change and when it is going to change you can never design a reliable circuit the circuit which works properly. So the best way would be to make any changes in the input when the clock is low. So when the clock becomes high the values get absorbed or stored in the latch, they remain in the latch for use in the next circuit or subsequent stages and during that clock high period you never try to change the input and again when the clock goes low if you want you can change the input one more time. If you want a reason to change the input do it at that time. This is generally a good practice in any digital system design or sequential circuit design. And the good practice is not to change the input when the clock is high because when the clock is high it can immediately affect the output whereas when the clock is low it will wait for the next clock pulse and only then it will affect the output. How fast you can do it depends on the mechanism by which you can generate this pulse train and how fast I want to do it depends on how fast I want the circuit to work. It is a very slow process and there is no need to make it very fast unnecessarily, for a process that is very fast it has to be done very fast. So this speed of the clock is not issue here. Of course there are other considerations as to what is the minimum value the clock can take and what is the maximum value that the clock can take. The maximum value the clock can take is the amount of delay that is required for the last to process your input change.

If you try to put an input into the latch before it can process and put it to the output if I try to change it one more time it is not going to be good, again it is going to lead to uncertainty output. So the minimum period for which you allow the clock to remain high is the period for which the latch requires to process the inputs. Therefore we talk about the propagation delay.

The propagation delay of the flip-flop is the time it requires to process. So, if you try to change your clock too frequently or too fast and change the input also then you will get confused. That is the highest speed I can work with. For lowest speed there is no limit of course there are some technology limitations.

Now there are some technology limitations in the lower frequency speed but you can operate it as slow as possible as slow as you require generally without taking into those extra technological reasons. So such a clocked latch is also called a flip-flop. The name given to this is called flip-flop and incidentally we have removed that sigma of 1 1 being the memory state and 0 0 being the unreliable state and 0 1 being the set state and 1 0 being the reset state now we want 0 1 to be reset state and 1 0 to be set state, 0 0 to be memory state, 1 1 to be not used state that has come because of this NAND gate here.

(Refer Slide Time: 20:58)



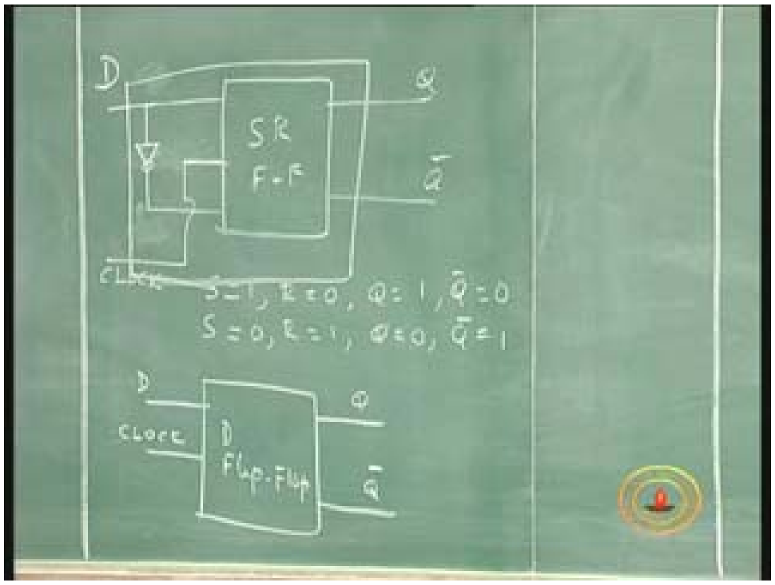
CLK	S	R	Q	$\bar{Q}$
0	x	x	memory	
1	0	0	memory	
1	0	1	0	1
1	1	0	1	0
1	1	1	not used	

If I put an AND gate it will still be the original thing. Since I put a NAND gate it has taken care of that problem, so this is a clocked SR latch also called as flip-flop SR flip-flop. Why is it called flip-flop is because the output changes from 1 to 0 0 to 1 all the time, only two states, it is like a swing, a seesaw either this way or this way, flip-flop.

Now this basic building block is not a latch that means it’s a clocked latch or a flip-flop, that’s the basic storage element. I want a clocked control to know when to put the data in and when to change it. Now let us look at the other problem. There are two more problems. I said the data variation is one, the second thing is in order to put 1 into this I need to put 1 0, S is equal to 1 R is equal to 0 if you want to put 0 into this I have to put 0 and 1 and someone asked me can I make it both at the same time S and R? It is better to do one at a time it depends on the circuitry you know. but the problem in this is since the clock is going to be controlling the whole operation the clock is going to be the time at which the circuit is going to start working it doesn’t matter when you put S and when you put R as long both are before the clock becomes high.

Still it is very inconvenient to store a bit of one I need to put 1 and 0 store a bit of 0 I need to put 0 and 1 I can easily remove this problem by making one of them always complement of the other. S and R are always complementary of the other because I want to put 1 and 0. So if you make a simple modification to this SR so now I will use this SR flip-flop instead of redrawing the whole circuit when I say SR flip-flop I mean this block (Refer Slide Time: 23:15) which has an S input, R input, clock input, Q output and Q bar output.

(Refer Slide Time 27:29)



If I now connect an inverter to this such that R is always complement of S so I can either have 0 1 as the input or 1 0 as the input. If S is equal to 1 R becomes 0 so Q is 1 Q bar is is equal to 0. S is equal to 0 R is equal to 1 I will make Q is equal to0 Q bar is equal to 1. So you want to set the flip-flop I put 1 in this, 1 here sets the flip-flop and 0 here resets the flip-flop. Earlier I could not have done it without the clock because now I can use the clock for memory operation. For a memory state earlier we have to put 0 0 and once I put 1 0 then I remove it and put 0 0 for memory or put 0 1 and I remove it or then put 0 0 for memory but now there is no need to put a separate 0 0 for memory. If I give a clock the input gets stored and if I remove the clock the input remains stored.

To store the input I give the input and enable the clock, to keep it all I have to do is to remove the clock make the clock low. So I put the value and keep it for the one clock cycle, and for next clock cycle if I want the same value I put the same value again and if I want a different value I put a different value. So this is the simple flip-flop where I have removed the condition of S is equal to 1 the two inputs being required to store one bit of information so this need not be then called SR but I can as well call it as a D flip-flop D for data. It stores a data bit a bit of data is 0 or 1 so my D flip-flop is nothing but a simple input called D and a clock Q and Q bar.

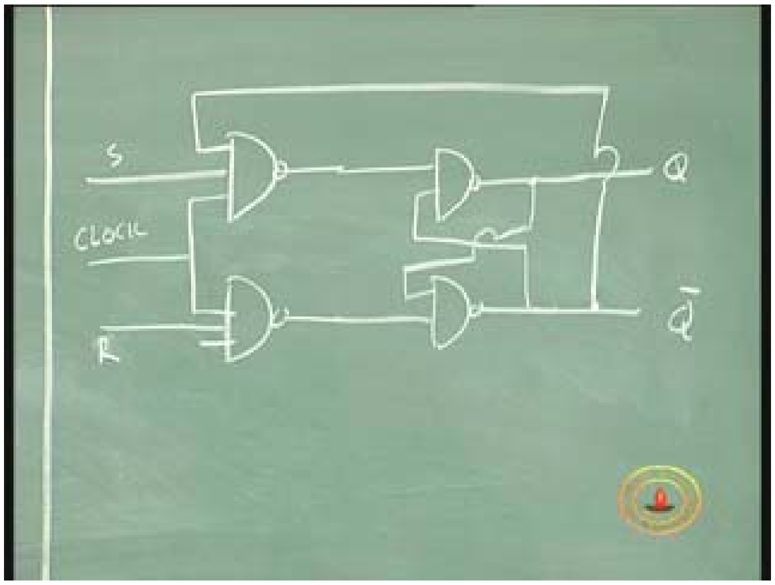
That means I have pushed this and inside this is an S and an inverter and R all that I am not showing what is shown here is this and this (Refer Slide Time: 26:35). I pushed that inverter inside the circuit and then it becomes a 1 bar. Incidentally we don't have to worry about the condition of S and R being 1 because just because of the inversion it is never possible to make S is equal to 1 R is equal to 1 so I don't have to worry about the non-permitted condition at all, it is a very frequently used flip-flop. The data storage is always mostly done in using D flip-flop so all you need to put is data and store it, put a data and clock it then it will get stored and if you remove the clock it stays. If you want to read it you can always read it any time because even without the clock the Q remains 1. If you want to change the data I need to put the clock again put the new value of the data. Therefore D flip-flop is a very simple one bit storage which is most frequently used for data storage. When you say memory for example in computer or a calculator any circuit it consists of all minor, fine of course the memory is huge but individually finally as I said whatever is the size of the memory but finally it is the bits that are stored in a D flip-flop. So, D flip-flop are very frequently used as a data storage device or circuit and that is why it is called a D flip-flop where D stands for data.

Thus we have an SR latch controlled into SR flip-flop introducing a clocking. I told you the reason for clocking. when I said what is the need for two inputs when the clock is there to control I can as well have only one input called D input and make the other input complementary of the other and then you got a D flip-flop so I can put a data store it and read it any time and if you want to change the data you put the clock again in the same place.

But when you look back at this SR flip-flop there is some attraction in that. There are two inputs so we have four combinations, memory state, 0 1 state, 1 0 state the output I mean and then the not used state. there is a circuit in which something is not used if that can be used also usefully if I can remove that “not permitted” stigma attached to the SR flip-flop and one more also can be made as the useful input in which some activity can be achieved some functionality can be achieved it will be even better so D flip-flop is ok for storing data but I want to now look at the SR flip-flop one more time but what is that we can do in order to remove that stigma attached to SR being 1 1.

So let us revisit SR flip-flop. In this case since I have to explain this operation I am putting the gate circuit again otherwise when I say SR I put a box and you know what is inside. When I say D and put a box you know what is inside. So in this case since I have to explain the functionality I am putting the circuit again using gates.

(Refer Slide Time 30:51)

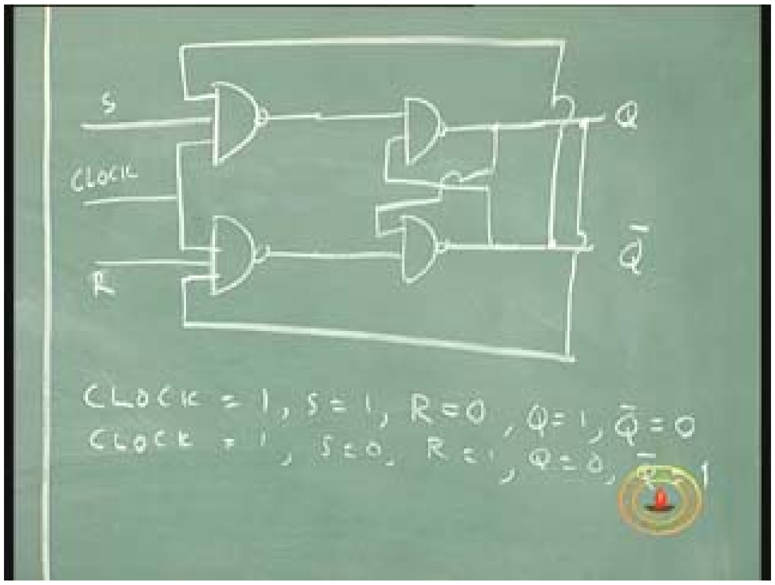


We call this clock (Refer Slide Time: 30:48) if you remember we wanted it as S R and you know how it works we just now saw a while ago how this circuit works. It is SR flip-flop provided you make sure that S is equal to 1 R is equal to 1 condition is not encountered, it will work as a normal flip-flop 1 1 storing 1 that if you put S is equal to 1 R is equal to 0 and storing 0 when you put S is equal to 0 R is equal to 1 and put 0 0 it will store whatever, it will keep whatever you have stored now the clock is an extra thing and with the clock only it works and without the clock again it remains in storage. So there are two ways in which you can keep what you have. one way is to remove the clock when there is no clock the clock is low and when you say remove the clock don’t physically disconnect the wire I mean clock 0 clock is not put on but the clock is off or put 0 0 when the clock is high and when you put input 0 0 still you can stay in memory state.

Now I am trying to make this into a three input NAND gate (Refer Slide Time: 31:50) I am going to connect the output of this into this I mean this input to this output and this input to this, let us see what happens to this change. Now I have taken the SR flip-flop and replaced the two input NAND gate by three input NAND gates of the input of the first stage and second stage remains as it is so 1 1 0 1 etc.

Now let us say if we are looking at clock high, clock low of course remains in memory state no problem so we will only worry about when clock is high. When clock is high or clock is 1 let us assume now S is equal to 1 R is equal to 0. When S is equal to 1 R is equal to 0 clock is high the output is going to be without this third input of the NAND gate and Q would be 1 and Q bar will be 0. And when clock is 1 S is equal to 0 R is equal to 1 again Q become 0 so this is normal I am trying to say that this is same as the previous thing, no problem.

(Refer Slide Time: 33:50)


S
CLOCK
R
Q
$\bar{Q}$
CLOCK = 1, S = 1, R = 0, Q = 1, $\bar{Q}$ = 0
CLOCK = 1, S = 0, R = 1, Q = 0, $\bar{Q}$ = 1

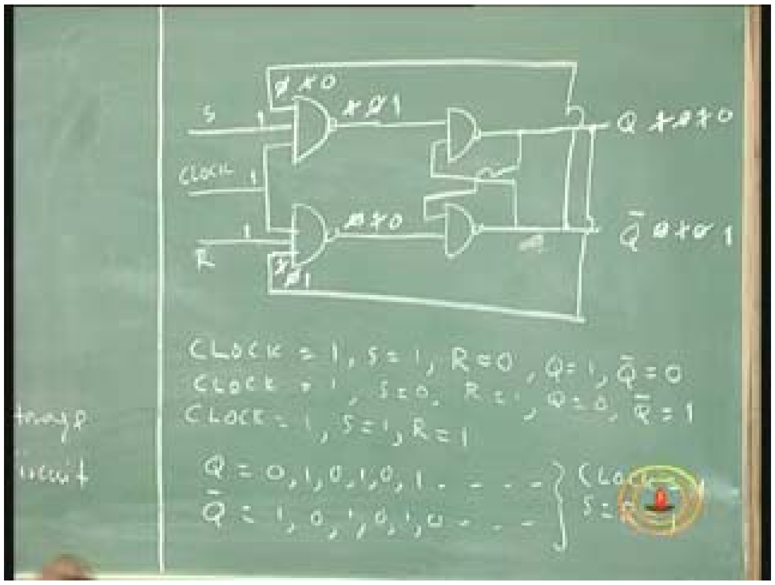
So at clock 0 state or 0 0 state they work as memory states S is equal to 1 R is equal to 0 also works as a set operation S is equal to 0 R is equal to 1 works as reset operation no problem with them. The only one condition which has not been dealt with the SR flip-flop we will now look at it. That is when clock is high S is equal to 1 and R is equal to 1 we will try to venture these input combinations. We will try to put these inputs combination and see what happens. Now clock is 1 S is equal to 1 and one of them is high and other one is low to start with. So let us assume that to start with this is 1 so we have to assume initially this is 1 and this is 0, this is 1 this is 0 (Refer Slide Time: 34:56) this 0 is here this one is here now we put clock is high R is high so all the three inputs are high for this, and here it is 0 1 1, 0 1 1 as the input here this is 1 and this is 1.

So I am making clock high S is equal to high R is equal to high. I am connecting back the output Q bar into this gate this 0 gets reflected here and 1 gets reflected here and 1 1 1 is the input of the NAND gate so what is the output? A 1 1 1 input gives you 0 and 1 1 0 gives you 1. And 1 0 at this stage what is the output? It is 0 1. So when you give 1 0 this one gets changed into 0 and 0 gets changed to 1.

I am putting an extra input to the two input NAND gates keeping S is equal to 1 R is equal to 1 keeping clock high and I am connecting simply Q back to the first R of this gate and Q bar back to this gate and I am now analyzing the performance. You have to arbitrarily assume. I can assume that first to start with that this was 0 and this was 1 as I have to start somewhere in the analysis so I assume this way, this Q is 1 and Q bar is 0 this one gets reflected here 1 1 1 gets an output 0 and 0 1 1 gets an output 1 so 1 0 at the input of the NAND gate, originally SR latch, remember that NAND 1 0 was a reset state, 1 0 is a reset state which was 0 1 and now this 0 gets back to this changing this 1 to 0 and this one changes from 0 to 1 now this NAND gate will have 1 1 1 input this NAND gate will have one 1 0 input so the output of this NAND will become 0, output of this NAND gate will become 1. When this 0 1 combination applied to this back to back latch back to back NAND gates the output would be 1 0, you get the drift.

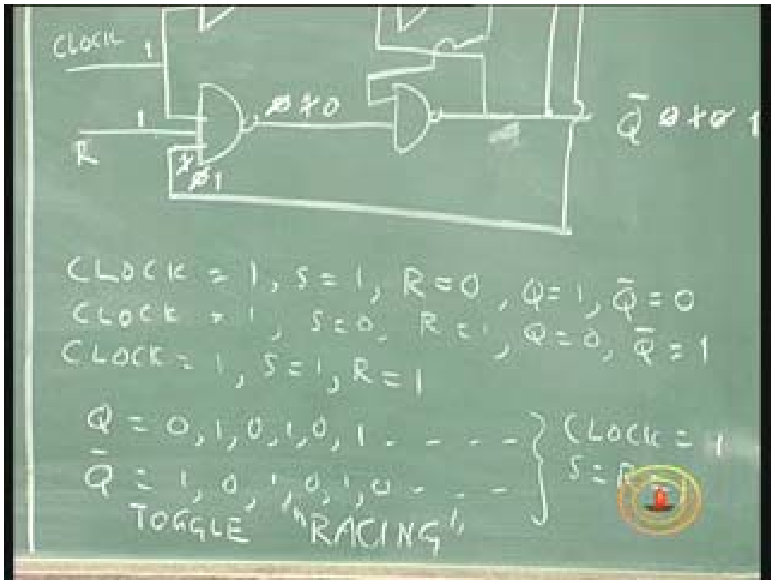
What I am trying to say is because this is the feedback as long as you keep the clock high and as long as they keep my S is equal to 1 R is equal to 1 all inputs to the NAND gates are frozen except the ones which are coming back the feedback inputs, those inputs were 0 1 to start with so when you put 0 1 this was 0 and this was 1 and the output is 1 and this is 0 so that 1 0 changed the output from 1 0 to 0 1 and this 0 1 got fed back made the inputs here 0 1 which made the output 1 0 I can go on repeating this. So if you now get this one back this becomes 1 this become 0 and this makes it 1, this makes it 0 so 1 0 here makes it 0 1 and it keeps going.

(Refer Slide Time: 39:17)


S
CLOCK
R
Q
$\bar{Q}$
CLOCK = 1, S = 1, R = 0, Q = 1, $\bar{Q}$ = 0
CLOCK = 1, S = 0, R = 1, Q = 0, $\bar{Q}$ = 1
CLOCK = 1, S = 1, R = 1
Q = 0, 1, 0, 1, 0, 1 . . . .
$\bar{Q}$ = 1, 0, 1, 0, 1, 0 . . . .

As long as I keep my clock high and as long as keep my R is equal to 1 and S is equal to 1 the Q changes from 0, 1 to 0, 1 to 0, 1 and Q bar changes from 1, 0 to 1, 0 to 1, 0 of course I am assuming clock is high S is equal to 0 R is equal to 1 clock is 1. A forbidden state of S is equal to 1 R is equal to 1 has been now converted into state in which the output keeps changing all the time. This very constant change of outputs is called toggling, this is the toggle action. This Q and Q bar changing uncontrollably if you are going to call it uncontrollably. The only way to stop doing this is to make the clock 0 or make changes in S and R.

(Refer Slide Time: 40: 35)


CLOCK 1
R
CLOCK = 1, S = 1, R = 0, Q = 1, Q̄ = 0
CLOCK = 1, S = 0, R = 1, Q = 0, Q̄ = 1
CLOCK = 1, S = 1, R = 1
Q = 0, 1, 0, 1, 0, 1 . . . .
Q̄ = 1, 0, 1, 0, 1, 0 . . . .
CLOCK = 1
TOGGLE
"RACING"

When S and R is in the other three combinations 0 0, 0 1, 1 0 it is no problem it behaves exactly like an SR flip-flop but the moment you put 1 1, in the previous case it becomes an operation which is not predictable and in the SR case whereas now we can say that it is predictable but useless. The output keeps changing from 0, 1 to 0, 1 to 0, 1 all the time it keeps changing but it may be useful sometimes, keeps changing all the time. This changing constantly is called a toggling operation. In fact it is changing so uncontrollably which we call as racing, it races. Toggle is only changing from 1 to 0 0 to 1 just like a seesaw as I said. If 1 become 0 and 0 becomes 1 that operation is called toggle. Actually strictly speaking is not a toggle it is racing, it is a race, something like oscillation where the output keeps changing from 0 to 1 to 0 to 1 to 0 to 1 to 1 and in the other case it is the complementary operation.

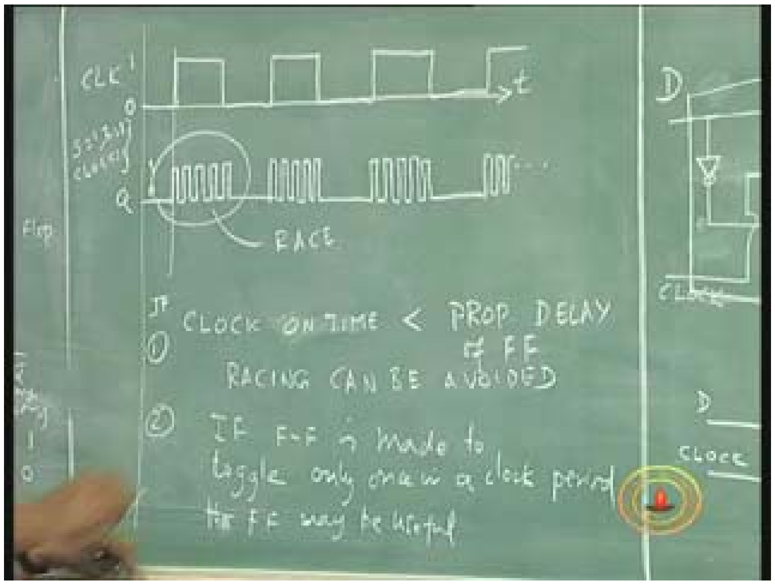
So if you want to draw the waveform for this output of flip-flop with S is equal to 1, R is equal to 1, clock is equal to 1 under that condition I am drawing the waveform for Q. when the clock is 0 there is no problem, the output remains let us say 0, Q is to be 1 as it is a memory state, we don't know if it is 0 or 1 but since we have to assume something let us say it is 0 but then it becomes 1, becomes 0 and then again it will even come down when the clock is 0 and again as the clock goes high it goes (this ….42:09).

Look at the idea, I keep on doing it so between state 0 and state 1 it keeps changing all the time. If it is 0 it remains 0 and if it is 1 it is not necessary it has to remain 0 it can remain as 1 also and the Q bar is a complement of that I don't want to draw that. This behavior …. race toggle is a mechanism by which the output changes from 1 output becomes 0 and 0 becomes 1 constantly but in a controlled way, but race is an uncontrolled way, the difference between toggle and race is race is uncontrolled and toggle is controlled when you want it then it changes back and forth all the time.

If you can control this phenomenon you can use it whenever you want an output to keep changing. We can use this and we will see where we can use this later on. But you can take it from me now that if you can control it so that it can only change once in a clock cycle.Tthis will be more useful if this clock cycle is 0 the next clock cycle is 1 then it becomes 0 then becomes 1 then becomes 0 1 0 1 and so on.

If it changes this, this pattern is interesting, I like to have this pattern but not as frequently like this but once every clock period or once every half clock period. When clock is not there it will become 0 and something like that. It is more controlled where I can predict the behavior. I said anything which is not predictable is not useful in any practical circuit. I cannot design a circuit and sell it to a customer and say you try your lucky with work for you. Do you like to sell something like that or would you like to buy something like that, no and that is why I don't want this, this racing has to be avoided. But now let us see how fast can it change, what determines the speed at which it changes? The propagation delay from here to here, moment I make this change at the input it takes a couple of clock periods I mean the gate delays to go to the output.

(Refer Slide Time: 49:46)


CLK
1
0
t
Q
RACE
CLOCK ON TIME < PROP DELAY of F.F
RACING CAN BE AVOIDED
IF F-F is made to toggle only once in a clock period the FF may be useful
D
CLOCK

So the propagation delay of the flip-flop determines the duration for which the output can remain constant and after that it has to change. So the speed at which it will change depends on the propagation delay of the gates. May be it is too small for us, one nano second, two nano seconds, let us say couple of nano seconds for this gate another two nano seconds for this, let us say two to three nano seconds for a gate a total of five nano seconds let us say. Every five nano seconds the output changes from 1 to 0 I can't do much with this. So I would like to have it changed but it a more orderly manner and at more regulated intervals.

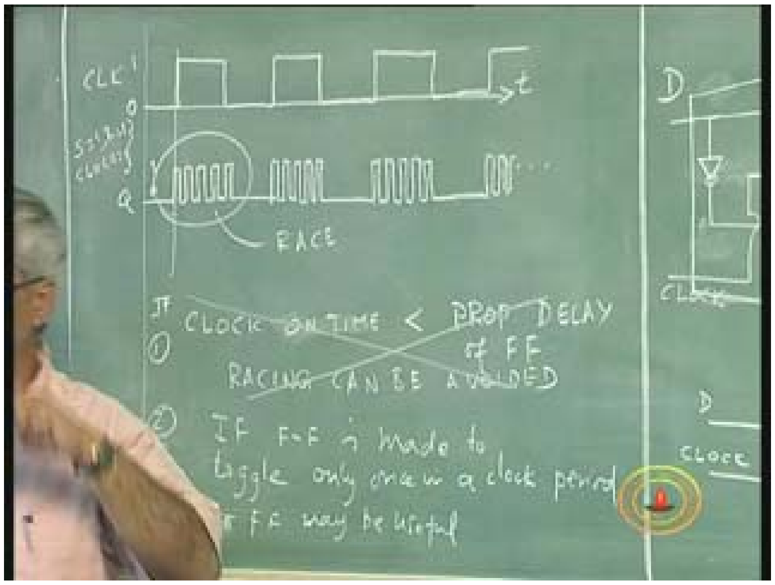
There are two things we can do. If I make a clock even smaller than this propagation delay I can make it only change once in a clock period. Suppose I give my clock, I said as well as the clock is high this is going to keep racing and when the clock settles down to 0 it will remain with the last value and again when the clock goes high it goes bizarre I told you that. So if I make the clock on time then it is less than the propagation delay of FF then racing can be avoided.

Can I have a clock which is working of course if it is necessary I have to do it but not for everything? For example I want to count people coming in to the room I want to have a very slow clock, many events are very slow, don't always think of Pentium IV which works with 2 Giga hertz even there lots of things are inside which is very slow, don't get carried away by the speed when the sales person tells you. What can you do with 2 Giga hertz per sec? Can you read something with that speed, no, can you input something with this speed? No, so all these are Miss Norman of course there is something which works inside which is 2 Giga hertz per second I mean but that is not what we really want for interfacing. So this is not a practical condition.

If I make my clock period very very narrow it will be less than this, if I make it less than this and go back to 0 if I can produce or generate a clock like that and use it for my flip-flop then I can have a condition that when the clock is 0 it will remain constant and when the clock is 1 it will become 1 and back and forth it will go. The other condition is somehow make the change possible only once in a clock period. That is what I am mean by toggling operation I would like the change of output from the present state to the complementary state. If the output is 0 I want it to be 1 and if the output is 1 I want it to be 0 so this is one possibility.

And the second possibility is if flip-flop is made to toggle that is why I use the word toggle in our 0 to 1 in a controlled way, toggle only once in a clock period race because toggle is a useful operation where it changes from 1 to 0 however large this clock period is, however long the clock period, whether it is a milli seconds, micro seconds, nano seconds or even seconds. When I have one clock pulse I will only have this change once not again, this will be useful. This flip-flop may be useful we will have to see how to do it.

(Refer Slide Time: 49:49)


CLK
0
STABLE CLOCK
Q
RACE
t
JF
① CLOCK ON TIME < PROP DELAY of FF
RACING CAN BE AVOIDED
② IF F-F is made to toggle only once in a clock period
JF FF may be useful
D
CLOCK

Thus what we have to see is, this is not possible, it would be an impossible condition to meet (Refer Slide time: 49:50). How to prevent this toggling or racing and make it toggle? In that case I have one extra feature of my SR flip-flop. SR flip-flop will work as it is, there is no problem other than the 1 1 condition it is a good flip-flop, it is not a bad flip-flop but it is a good flip-flop. That means it has 0 0 state and at clock 0 it behaves as a memory and with 0 0 it behaves like a memory, with 0 1 it is reset and at 1 0 it sets, these three features are common but 1 1 condition is a […….50:25] condition which is not there I am trying to term it in that process I have got a beast which is race I have to remove that [beast] operation and make it a useful operation, how to do that we will see the next lecture.