Digital System Design

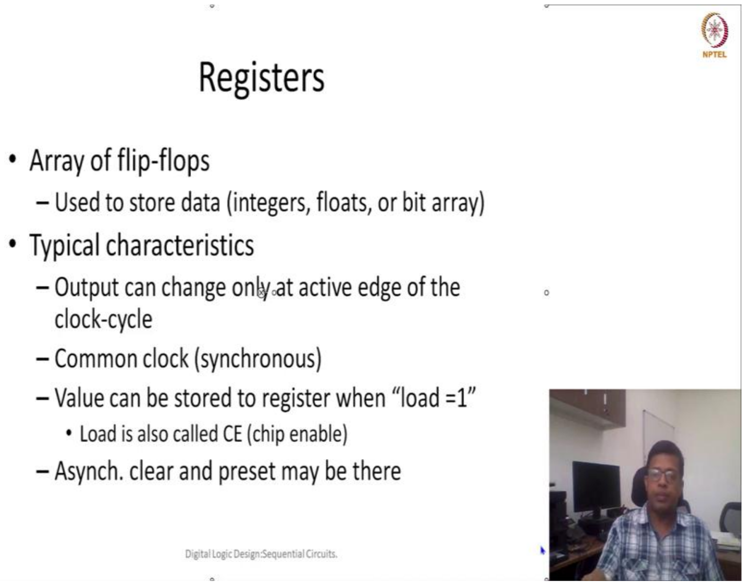
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology Ropar

Lecture 47

Registers - 1

Hello, all. Today we are going to see another application of flip-flops. These also are very standard component in sequential circuit. These are call Registers.

(Refer Slide Time 00:30)



Registers

- Array of flip-flops
  - Used to store data (integers, floats, or bit array)
- Typical characteristics
  - Output can change only at active edge of the clock-cycle
  - Common clock (synchronous)
  - Value can be stored to register when "load =1"
    - Load is also called CE (chip enable)
  - Asynch. clear and preset may be there

Digital Logic Design:Sequential Circuits.

Now, let us understand first what is a Register. A register is usually an array of flip-flops. So that means multiple flip-flops are together will form a Register. Now, as flip-flop is used to store single bit of data, because it is an array of flip-flop, it would be used to store multiple bits of data. So for example, it could be, these multiple bits could be structured or unstructured.

Structured means it could be integers, float or character and in terms of unstructured, so it could be like group of bits which are used for one particular purpose. It could be a control bit, it could be a data bit. So, when we are using multiple bits, the purpose is actually to store. So store the data. It could be for multiple clock cycles, it could be for definite amount of time, etcetera.

Now, because these Registers, because the Registers are actually the storing elements, the elements which would be used to store the data. Now, there has to be some typical characteristic so something which, as soon somebody says Register, what do we recognize? One thing is that their output will change only at the active edges of the clock cycle. So what does it mean?

Let us say the active edge is a positive edge. So the output can change only at the positive edge of the clock cycle. So the side effect of this particular characteristic is that your output cannot change in between. In other words, your output would be stable for one clock period.

Now, the other side effect is, let us say, your input is coming from a register so that means input will also not change during one clock cycle. So this particular characteristic is in common with flip-flops but here because we are storing whole amount of data, integers or characters or any other form of data, in Registers, so this become more valuable that your output does not change for one clock cycle.

The other thing that, Registers are synchronous. Basically if they are array of bits, all of these bits will flip at the same time. So, if there are array of flip-flops, all of the flip-flop would be driven by same clock cycle, same clock signal. So because they are driven by same clock signal, they will change at the same time. So that is why it is called synchronous.

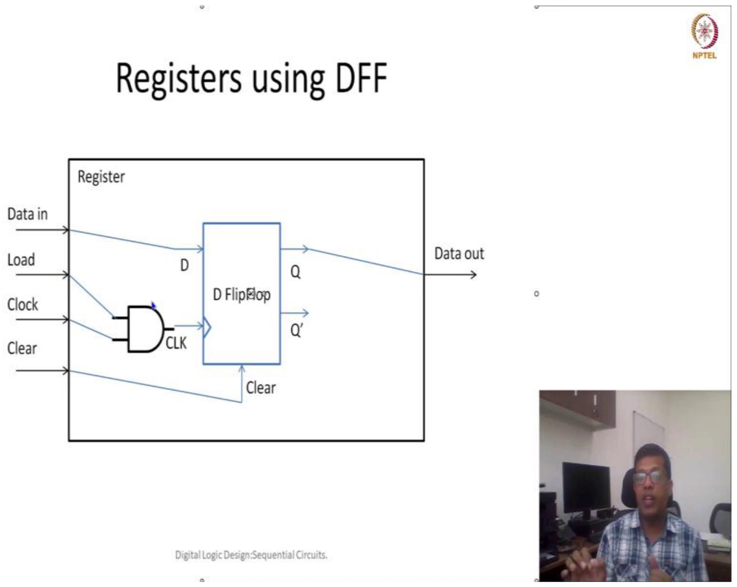
Similarly, they should also be synchronous with other Registers inside the design. So in the design, there could be multiple Registers, all of them are supposed to change with the same clock cycle, same clock signal. The one thing which differentiates Registers from flip-flops, so in flip-flops, what happens is that as soon as you give a input and based on the clock signal, at every edge, at every active edge of the clock signal, you will sense what is the input.

Now, in Registers also we do the same but if there would be an additional signal like chip enable or clock enable or load signal, so when that load signal or load signal is 1, or CE, it could be, sometime people call it chip enable or sometime peope call is clock enable, so this CE signal or load signal, when it is 1, then only data will be sensed at the active edge of the clock and then it would be stored in the Register for the minimum amount of time.

If load is 0, that means, data will not be changed. Data values inside the register will not change and they will remain same. So in other words, if I want to change, if I want to store something in my register, then I have to, I have to give my data and also, I have to make my load signal 1, or CE signal 1 at correct time so that when active edge of the clock will come, then it will see that load signal is 1 and whatever data is there at that time, it will sense it and it will store it in the register content.

In addition to load signal, there could also be an asynchronous clear or preset signal so because these Registers, they would be utilized, they would be utilized and reused during the whole design. So we have to initialize something, the Register or something, some value. So that is why, asynchronous reset, asynchronous clear or asynchronous preset, these signals may also be there.

(Refer Slide Time 05:31)


Registers using DFF

So now, how do we design, how do we design such kind of a register which are a memory element, which has these characteristics that they are, they will change output only at the active edge and they do not change their input when the load signal is not there. So we can see that because most of the characteristics are actually overlapping with D flip-flop, so we can use D flip-flop to design such a register.

So now, we have seen that the characteristics these are the input, we have a data input, we have a load, we have a clock signal and clear and data out is the output here. So these are the input output controls of my Register, a typical Register. Now, if I want to use D flip-flop to design them, then what I can do is, in my D flip-flop, these are the typical input output. I have D as an input, clock as input and clear is a asynchronous input, Q and Q r could be output.

So one thing I can do is, I can connect my D into D, so I could connect my D into D, I can connect my clock to clock and clear to clear and D out to, Q to D out. So that means essentially, it looks like there is no difference between Register and D flip-flop. Correct. But we have an additional characteristic in the register that when this load signal should be 1, then only data should be taken in. Otherwise data, my output will, should not change. My Q should not change, it should remain as whatever I have stored it earlier.

So, that means this design will not work, I have to do something smarter. One thing I can do is, I can remove this clock and I can connect this load signal and this clock signal to a AND gate and then the output can be connected to the clock of the D flip-flop. This is also workable solution but you see, your clock, so if your load is 1, then only the clock would be enable, if load is 0, clock is not given.

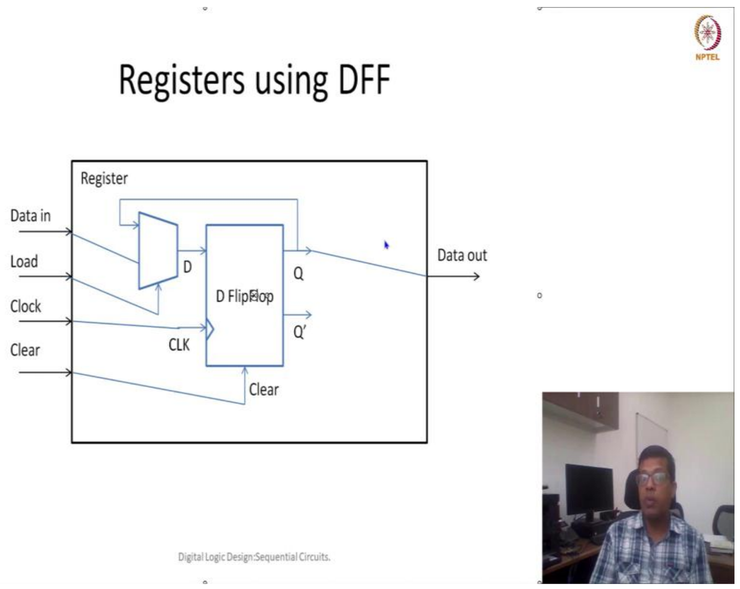
So, that means whatever we want to give as a data in that can also be sensed when the load is 1. So this, theoretically this design will work. There is no issue, theoretically with this design, at least with the specification what we said but there is two specifications which is not matching. One thing, that this particular design is not a synchronous design. So when we were saying synchronous, synchronous means that all the Registers in the logic, they will change at the same time.

Now, since my clock has been modified, now this clock, which is given to my D flip-flop, will be affected by this AND gate. Because of the timing of the AND gate, this clock would be delayed than the input clock. So would be the output. So output would also be, will be synchronous with this clock which is a delayed one, so that means this particular Register which is, where we have modified the clock cannot be synchronous with all other Registers in the design, all other flip-flops in the design.

So, that is the, that is one issue. The other issue is also there that because, that is a side effect of the load signal, so let us say this load signal changes, sometimes. So my clock was 1 and during this clock is 1, load became 1 and then became 0, then it has artificially created one additional positive edge which is not desirable. So that means, it has created another synchronization point which is not otherwise existing.

So that is why, because of these two reasons, modification of the clock as well as because of this clock AND-ing with the clock, load and clock being AND gated, so because of that there could be some spurious activity, there could be additional positive clock edges. So, then data would be sensed at that time also. So because of these two reasons this particular design is not preferred. Instead, what we have to do, what we would like to do, we would not like to change my clock signal at all, my clock should be directly fed to my D flip-flop, without any additional delay.

(Refer Slide Time 10:23)

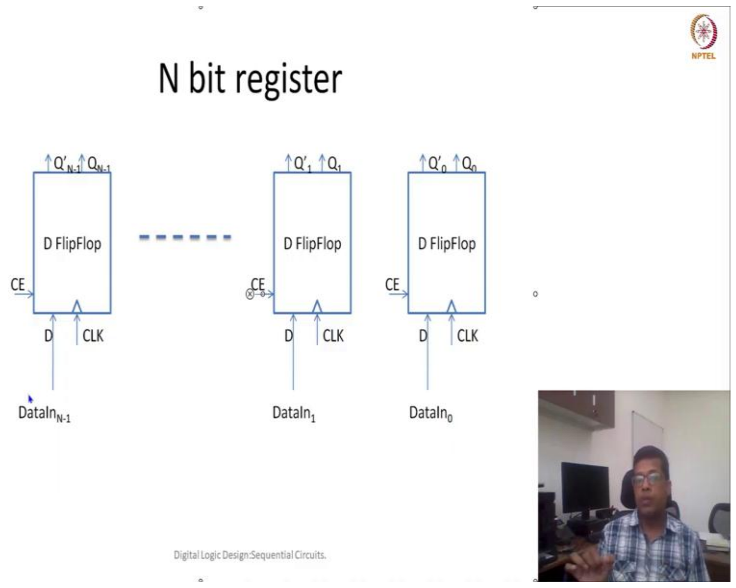


So, this is for sure, that clock has to be connected something like this, but then, I have to do in such a way that this load signal will be, will still be able to utilize in such a way that this data in can be applicable only when the load is 1. So what I can do is, I can possibly have a multiplexer here. So this multiplexer is an additional feedback loop. When load is 1, that means I would like to take data in as an input, then my multiplexer would select data in.

If my load is 0, then I will connect my data out signal and will give it back to D. So what this, this feedback loop will ensure? What this feedback loop will ensure that when load is 0, the output will not change because we are taking the output from the actual output of the D flip-flop so, here, input will also be same, whatever was the previous input, previous output. Because my input is same as the previous output, next output will also going to be same as previous output. Output will not change.

Output can change only when load is 1, so multiplexer will select data in and then output can change. So, that is how we have to implement a register where clock is not modified but we have modified our input to D flip-flop in such a way that the input will not, output will not get modified when load is 0. And when load is 1, then only data input would be taken and my output can change. So this particular implementation will take into account all the characteristics or all the requirements of my Register.

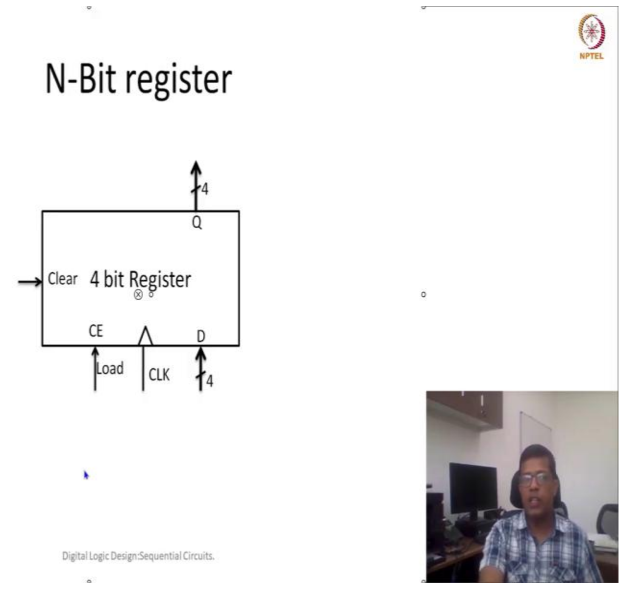
(Refer Slide Time 12:30)



N bit register

So this, this particular flip-flop is also, this particular Register is defined by this diagram where we gives, we instead of load, we are writing it as CE and then D is an input and clock is an input and Q and Q bar could be the output. So in N-bit register, essentially all of them would be, there would be an array of N D flip-flop. Each of them would be connected to clock enable or load signal and each will have their own data input signal, all of them are binded to same or connected to same clock signal. So outputs are different. D input signals are different, data in signals are input but all of them are connected to same clock enable, same clock signal.

(Refer Slide Time 13:23)

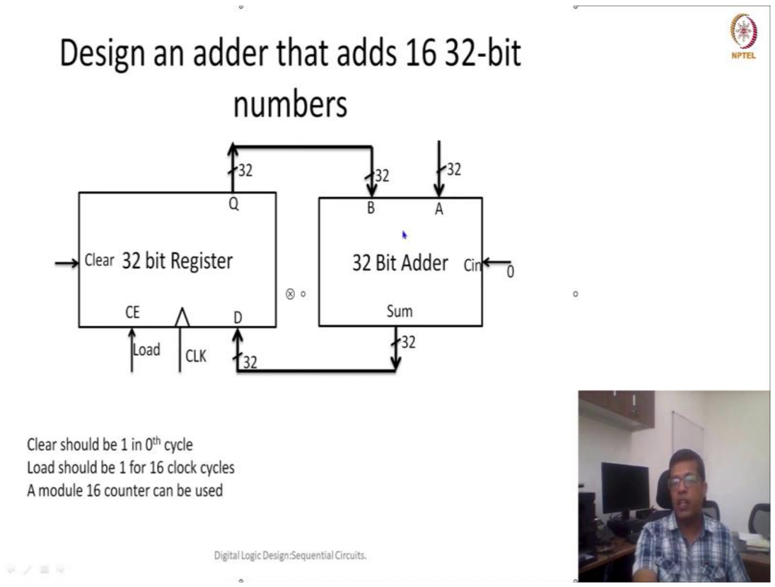


So I can represent this as one block diagram where I can say that, let us say, it could be an N-bit register. If it is a 4-bit register then D would be, input is four lines or basically a bus which has four wires and output is also bus with four wires. There would be a clock signal, there could be a load signal which internally would be connected to CE signal and there could be asynchronous clear, asynchronous preset. So this is how we typically represent an N-bit register.

So essentially, it would take N number of inputs, N number of outputs. This Register is also called parallel in parallel out Register because all the inputs are given in parallel and all the outputs are also received or can be read in parallel. So this is about Registers.

Now, let us quickly see, can we utilize or where do we utilize these Registers. So let us take one example or one application.

(Refer Slide Time 14:30)



Design an adder that adds 16 32-bit numbers

Clear should be 1 in 0th cycle
Load should be 1 for 16 clock cycles
A module 16 counter can be used

Digital Logic Design:Sequential Circuits.

So, the question which we have in hand is we would like to design an adder that ads 16 32-bit numbers. So you remember in your last module we have seen a good number of techniques, one of them was carry save adders. So these carry save adders can be used to design an adder which has multiple operates. Of course, we can use that technique.

Now, we have also seen that if we use that technique, the critical path, the critical path or basically the number of full adders which would be required would be of the order N. So here, if I have 32-bit numbers and 16 additions. So it could be like, it would be order of 32 full adders. So that would be delay of one, delay of such an adder. So essentially, in other words, what would happen that if I would use those carry save adder techniques, the delay of this addition would grow, would be more than 1 32-bit adder.

Yes, it is implicit also, or it is intuitive also that the time which is required to add 16 32-bit numbers is going to be more than adding 1 32-bit number. Additionally, it would require more number of full adders or in other words, more area would be required. So on the other extreme, we have also seen at the time, another design, where we were adding all of them sequentially. So there also we said that the total number of adders required are, 15 adders would be required.

Now, let us consider, another method. Here, that we take these 2 32-bit numbers and keep on adding and whatever is a result, we store it in a Register. And from the Register, we read it again and then add the another number. So if such a design is there, what we are seeing that we are using only 1 32-bit adder. So if we are using only 1 32 bit adder, then every time what we are doing is, we are adding 1 additional operant and whatever is the result, that sum, we, if we store in one Register and then the second number could be read from that register.

So essentially, it is a kind of an accumulator combination. What do we mean by accumulator? So let us say, we want to add 16 numbers a1 plus a2 plus a3 plus a4. So result of a1 plus a2 would be stored in a Register and in next clock cycle this result can be given to the next number, one of the operant and then we are adding a3. So this is already a1 plus a2 and now, a3 would be added, sum would again be stored in my Register.

And after the end of clock cycle, now, this output represents a1 plus a2 plus a3, now, then, I can add a4. So every clock cycle, we are accumulating one more number. So this way, we would be able to do all the arithmetic using 1 32-bit Register and 1 32-bit adder, where sum is being stored in a Register and from the Register, we are reading every new clock cycle and a new operant is being fetched every clock cycle.

So, the idea here is that now, who will make sure that all the different inputs of A are coming at right time and we also have to make sure that in the starting of computation we make clear equal to 0. So in the starting of the clock cycle we say clear equal to 0 so this 32-bit register is essentially 0 and after that it will start accumulating. And how many clock cycles would be required? Because in one clock cycle, we are adding one operant, then next operant, then next operant so a total of 16 clock cycles would be required.

So if somehow we are able to count those 16 clock cycles, we know that in initial or at zeroth we say that my Register is 0, at the end of 16th, my Register, this Register would be able to show the sum of all the 16 numbers. So now, similarly this 16, so essentially in this whole system, there would be another 16-bit counter, sorry, module of 16 counter or a 4-bit counter would be utilized which is incrementing again and again and which is making sure that end of 16th cycle, we know that result is available.

So there, this load is going to be 1 all the time, remain 1 all the time because every clock cycle we are adding to this, we are saying that Register should report. So after 16 clock cycles, we can either make my load as 0 so that means no further addition would be there or no further, this adder will not be utilized. So in other words, there is another way also that let us say, after 16 clock cycle, again we would like to start this adder then we can make this clear 0 and after 16 we say that yes, now, output is available, we can restart.

So all those permutations combinations are there but essential idea here is that yes, if we would like to simplify some of these designs, we can simplify by creating this kind of a circuit where we are able to, able to utilize the output. We are able to store the output in the next clock cycle, that output can be reutilized. So here at the end of the clock cycle, this output is taken as the input. So, this way, this input and this output is also segregated.

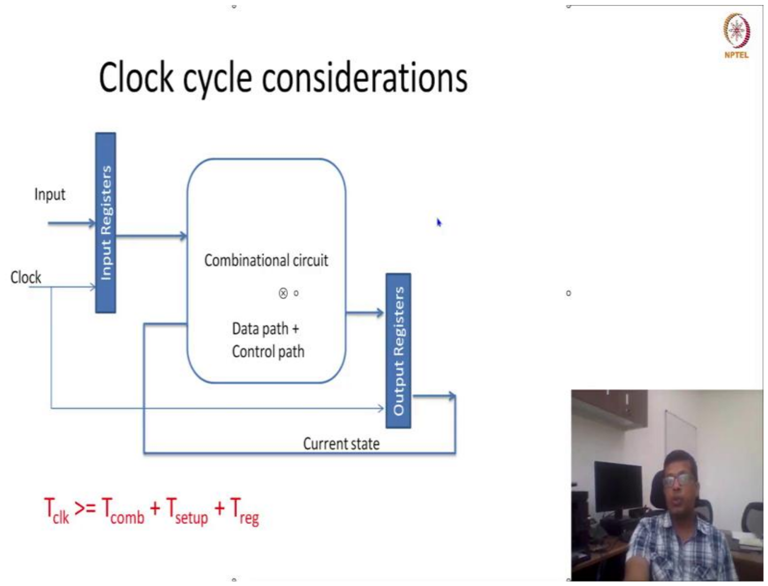
Now, whatever is available at the end of clock cycle here, before the active edge, will remain there for one full clock cycle in the output. So that means this B will not change for one clock cycle. Now, in this one clock cycle my sum will keep on computing, so you understand, you have done some of the experiments, you have seen that the, if we are using some ripple carry adder or even look ahead adder, my output will not be stable unless the latency of this particular adder is there or worst case delay is there we cannot make sure that my sum is stable.

But once sum is stable, once this output is stable then we can give it to D and then at the active edge, this D would be sensed and can be given to the output. So all of these things are giving us a sense that, that means we have to also make sure that my clock signal is also wide enough, my clock size, duration of the clock is appropriate that it is, it is able to handle the worst case path.

So, we will see that discussion. So the other point here is that, this particular example, it is okay if you are not able to understand completely but this example is to give you a motivation that these Registers can be used to certain computations which can be done in a phase wise manner. In one clock cycle, you do one step, in second clock cycle you do next step, etcetera.

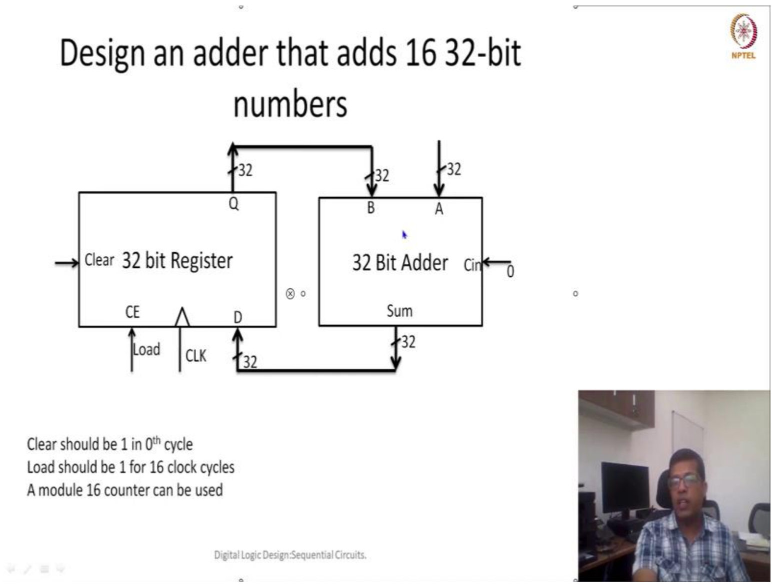
So in forthcoming modules, in the next module, we will take much more such examples and to see that how those computations can be done. But right now the more important point here is that yes, these Registers and flip-flops can be used to do computations which can be done in clock by clock manner or step by step manner. Now, coming to the clock period, so my clock period has to be wide enough that my addition is finished, my Registers delays are also incorporated.

(Refer Slide Time 23:35)



Clock cycle considerations

$T_{clk} >= T_{comb} + T_{setup} + T_{reg}$



Design an adder that adds 16 32-bit numbers

Clear should be 1 in 0th cycle
Load should be 1 for 16 clock cycles
A module 16 counter can be used

Digital Logic Design:Sequential Circuits.

So this diagram, now, summarize, like, the very generic combinational, very generic sequential circuits. So here, we are seeing that we are taking Registers as a input. So you see, this input is coming from a Register. B input is coming from a Register and the output of combinational circuit, here, the output of my combinational circuit sum is also being stored in a Register. Incidentally here, input Register and output Register both are same.

So, yeah, here is the case. So because this output Register is also given as input, so here my output Register and input Registers both are same. But the other input Register, so you have this A, is also coming, we have to assume that this A is also coming from some Register so that it does not change for one, during one clock cycle. So now, what should be the delay, what should be the clock period size? My clock period depends on the delay of combinational circuit, adder 32, in this case.

So this combinational circuit will have certain delay. Not only this, my Register will also have certain delay, my Register has, you remember, the three kinds of delays, one is setup time, hold time and propagation time. So setup time is the time for, like my input should be stable at least that much time before active edge of my clock.

So this, my output of combinational circuit should be stable at least setup time before active edge of the clock. So that means the total delay has to be combinational delay plus setup time. Additionally, the output of the register will not be available immediately. It would be available only after the propagation time of the Register so that I am calling as T reg. So this Register will also take some delays, some propagation time.

Now, that is the time your output would be available to work on. Or in other words, after active edge of the clock, my input to combination path would be available only after T reg amount of time. So clock period has to be more than combination delay, worst case combinational delay plus setup time of the Register plus propagation delay of the Register. So, that is, that should be the minimum clock period, if my clock period is lesser than that, we cannot guarantee that circuit will work correctly.

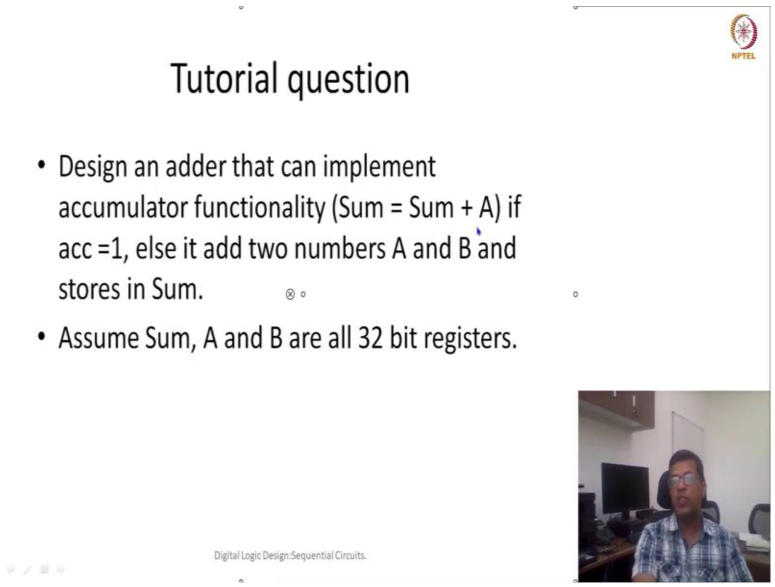
So now, you can ask one question here that what about hold time? Why hold time is not featuring in this equation? Now, hold time, there are two reasons. One, usually hold time is lesser than my propagation delay. The other reason is that my output, my input is available only after propagation delay from the positive edge of the clock so unless my input is available, my combinational circuit will not start working. So this T reg become in the critical path.

So then you can also ask, what is the reason, what is my T hold time is more than propagation delay? You see, my output has already changed. flip-flop output has already changed. What is the purpose of keeping my input till, still without any change? So, why should I hold my input to the same value, in my hold time, if my output has already modified? So, taking this as an intuitive idea, so hold time, theoretically cannot be more than the propagation delay.

So this should be the clock period and because of this whole thing, this will always determine, whenever we are trying to determine what is the clock period, so that would depends on the combinational path. Sometimes if there are multiple Registers, then we have to see, let us say, again, take this example, so let us say, there is also, the same Register is also doing a computation which is addition as well as multiplication. In that case, multiplication would be in critical, addition may not come in critical path.

So when we say combinational circuit delay, it would be multiplication delay, not the addition delay. So worst case delay of my combinational circuit between two Registers plus setup time plus propagation time would be the clock cycle. So this was about Registers or parallel in parallel out Registers or regular Registers which would be used in sequential circuits. Let us move on to a new problem.

(Refer Slide Time 29:11)



Tutorial question

- Design an adder that can implement accumulator functionality (Sum = Sum + A) if acc =1, else it add two numbers A and B and stores in Sum.
- Assume Sum, A and B are all 32 bit registers.

Digital Logic Design:Sequential Circuits.

So, before we move on, there is one question which you can take it as a homework. So instead of tutorial, let us call it a homework question. So you try to design an adder which is, which implements a accumulator functionality, conditionally. So what is an accumulator? In accumulator, you have, this equation signifies my accumulator, my, one of the input is also one of the output. So Sum is equal to Sum plus A.

So that means we have to assume Sum as well as A, both are stored in a Register. If they are stored in a Register, so as soon as Sum plus A is computed by end of clock cycle in the next clock cycle, it would be the new Sum. So we are saying that you have to design it but conditionally, let us say if accumulator is 1, then we have to do this functionality. If accumulator is 0, if acc equal to 0, then, these two numbers could be different A plus B.

So instead of Sum, there could be B and A could be same so, and then still, you have to store the result in Sum. So you try to design this, this circuit which is quite similar to the previous one but with some modification. Assume that A, B as well as Sum, all of them are stored in 32-bit Registers.