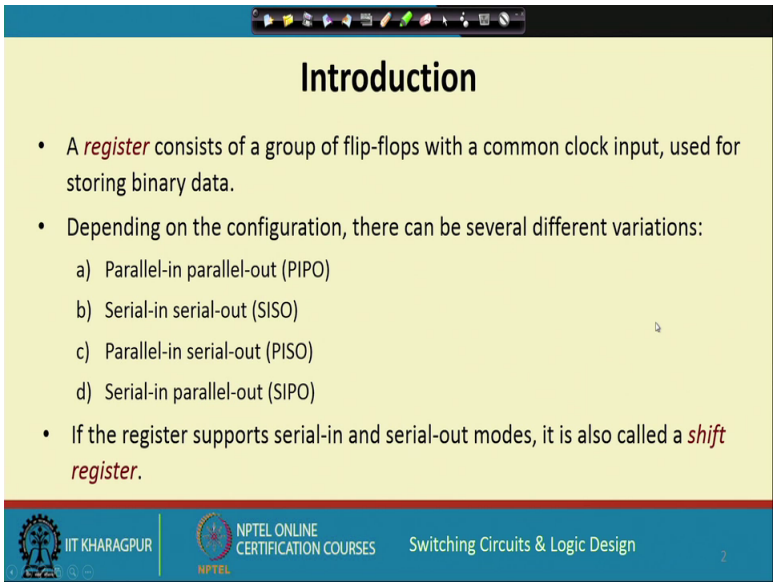
Switching Circuits and Logic Design
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 42
Design of Registers (Part - I)

So, in this lecture we start our discussion on the design of certain sequential circuit building blocks. Specifically we shall be looking at the design of registers and counters, the different types of registers that can be there, different types of counters and some typical applications. So, the title of today's lecture is Design of Registers the first part.

(Refer Slide Time: 00:47)



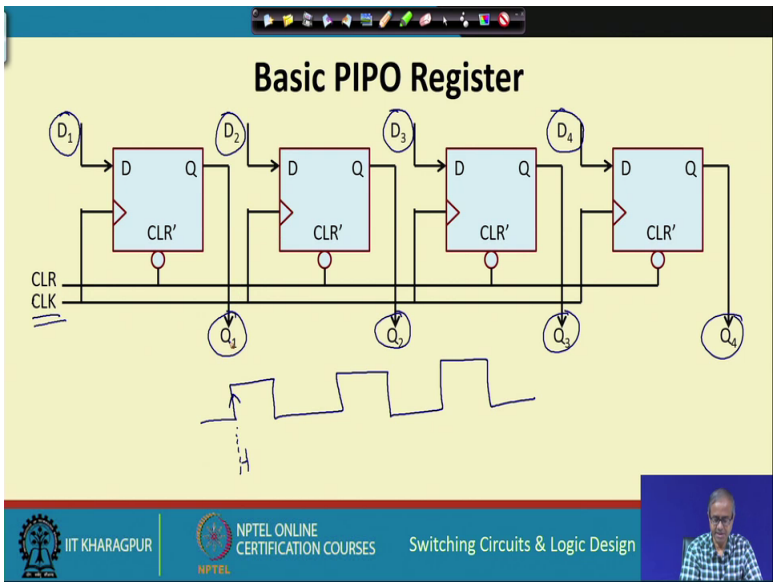
So, let us try to first understand what register basically means. Look we have seen what a flip flop is. A flip flop is a storage element which can store 1 bit of data, but there are many applications where we need to store not just 1 bit of data, but an entire word of data that word size can be anything it can be 8, 16, 32, 64 anything.

Suppose I need to store a 16 bit word then I can use 16 such flip flops to store all those 16 bits of data and we call it as a register. Register is nothing, but an array of flip flops, but how the flip flops are inter connected that distinguishes the different types of registers ok.

So, broadly speaking a register is a group of flip flops. Typically there is a common clock input there, is a clock input coming from outside which is feeding all the flip flops in that group and it is used for storing binary data. Now not only just storing depending on how you are connecting the flip flops, there can be several different variations of registers. Specifically registers can be of four different types depending on how you are using them parallel in parallel out, serial in serial out, parallel in serial out and serial in parallel out.

Now, we shall be discussing these variations in due course of time that what are the main difference and how we can design a register that support these features ok. Now just one thing to remember if the register is such that either or both of serial in and serial out modes are supported which means I am talking about one of these 3, then we call it a shift register. Shift register is a special kind of a register which supports either serial in or serial out or both of these fine. Now, let us look at these different types of registers one by one.

(Refer Slide Time: 03:24)

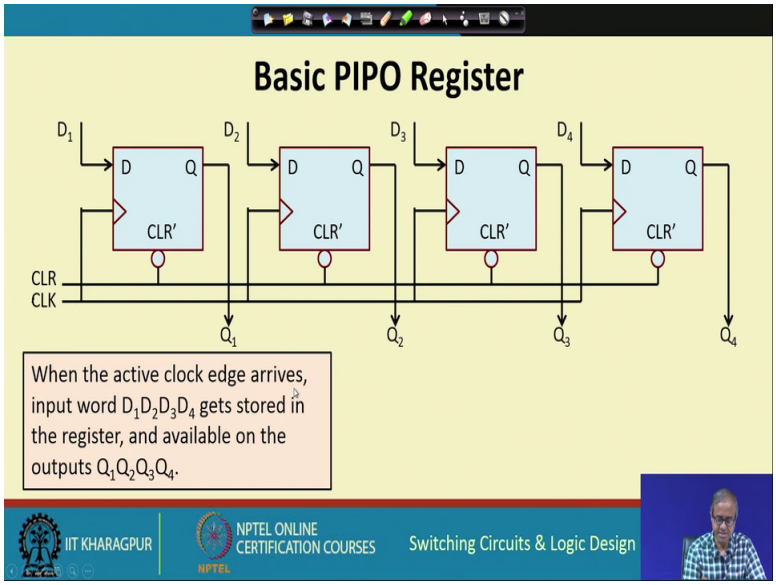


First the simplest one the parallel in parallel out register. I have a group of bits a word I just want to store that word in a register and when required, I want to read the value that is stored. So, as you can understand I can take an array of flip flops, D flip flop will be the simplest because in D flip flop whatever you give in the input after the clock comes that will get stored. So, if I have an array of D flip flops, I can very easily implement a parallel in parallel out or PIPO register.

So, the basic PIPO register is shown in this diagram. So, this is a four bit register which is shown you see there are four D flip flops we are using. The parallel data input which is coming from outside is fed to the D inputs of this flip flops. These are D 1 D 2 D 3 and D 4. Now as you can see the clock input is being fed in parallel to all the flip flops. So, when the clock comes, suppose in have a clock pulse coming like this. So, if it is positive which is triggered whenever the positive edge of the clock comes, whatever I have given in D 1 D 2 D 3 D 4 will get stored in the flip flop and they will be available on Q 1 Q 2 Q 3 and Q 4 this is how it works. Of course, after the clock comes there will be a small delay this delay will be determined by the propagation delay of this flip flop after that the output Q will be changing state whatever you are applying D 1 will go to Q 1 and so on.

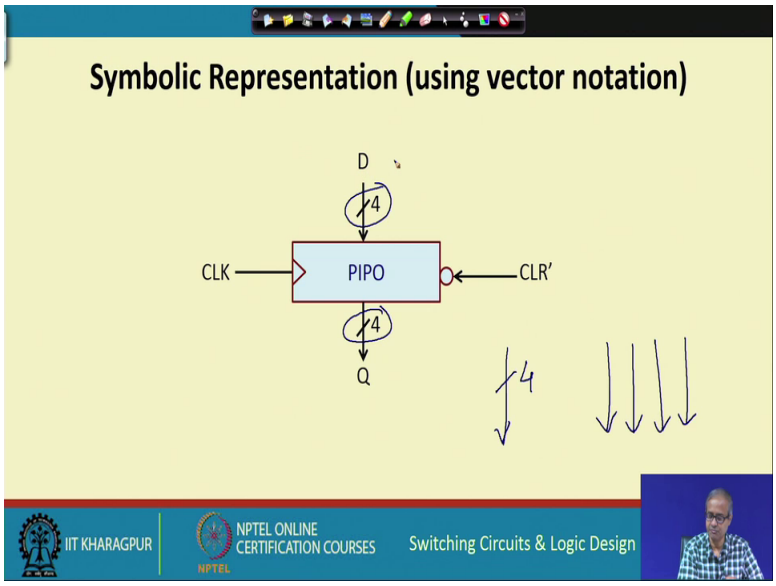
And this flip flop as you can see this also as a clear input, which is active low because this bubble that we are showing here this means it is active low active low means whenever clear is 0 then the flip flops will be cleared. So, if you connect this clear input to all the four flip flops. So, whenever you want to clear them to all zeros you can do it ok. So, the basic parallel in parallel out ship register works like this.

(Refer Slide Time: 06:00)



So, as I said that when the active clock edge arrives here it is leading edge, rising edge the input word this the four bit register, will get stored in the register and will be available on the outputs. So, the PIPO register is very simple in concept right.

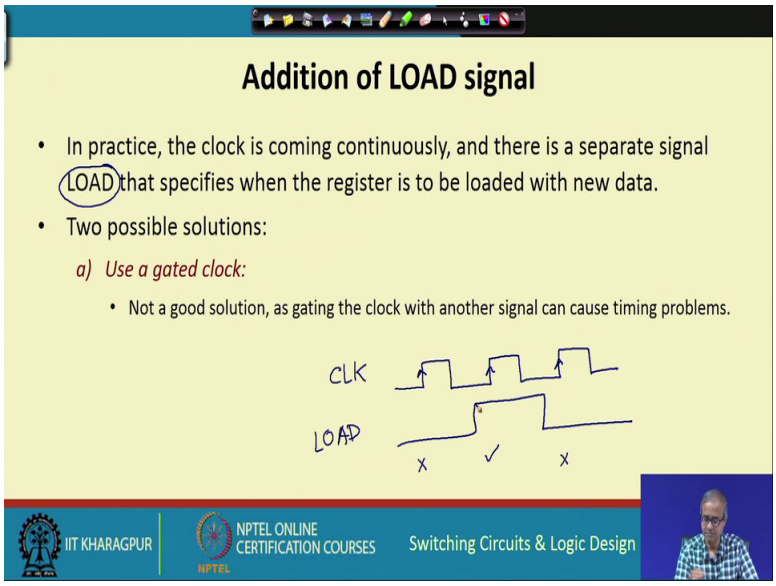
(Refer Slide Time: 06:23)



Now, a PIPO register can be represented symbolically like this we do not always have to draw all the four flip flops to show that it is a four bit register, we simply say that this is a PIPO register there is a clock input this symbol is clock, there is a clear input again active low symbol is there and D and Q are shown with vector notation. You see this four as you can see here. So, in this line we have made a cross and then we wrote 4 by the side of it. This means actually this is a collection of four different lines not 1.

So, the number of bits or number of lines I can just represent it like this and this is called vector notation. So, in vector notation D actually means D 1 D 2 D 3 D 4 there are four lines and Q means Q 1 Q 2 Q 3 Q 4 ok. So, this is how symbolically a PIPO register looks like.

(Refer Slide Time: 07:35)

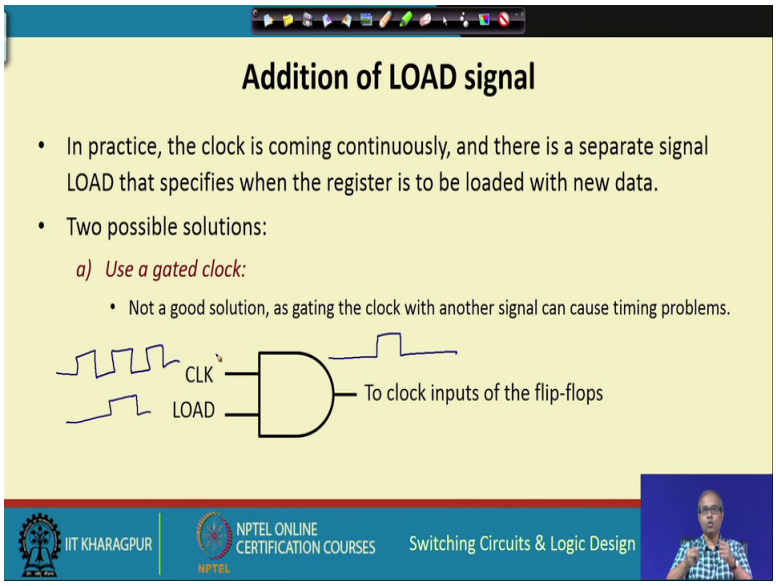


Now, let us add some features to this PIPO register. Now the register that we have seen so far there the data gets stored whenever clock comes. Now you know clock is a continuous signal it keeps coming, but all the time I may not be having some data to be stored in the register from outside. So, there should be some kind of control or facility of available to me as a designer. So, that I can say that well now you have to load the data let the clock come, but whenever I say load only then you load the data and at all other times do not disturb the content of the register whatever was stored let it be there ok.

So, here we are talking about the addition of a load signal. Load signal what you are saying is that we have a separate signal called load, the idea will be like this suppose the clock is coming let us say this is my clock, and these are the active edges of the clock rising edge and suppose this load signal I am giving like this, suppose I am giving it like this. So, when the first clock edge comes load is 0. So, there will be no loading; when this second clock edge comes load is high. So, there will be loading and when the third clock edge comes again load is 0 there will be no loading.

So, I can selectively specify when I have to load by suitably applying the load signal, but we remember the load has to be made high a little before the clock edge comes; because if you make them high together then it may not accept the high level of load alright. Now the first approach in which you can use the load signal you can implement the load signal that 2 approaches you shall see this is something using a gated clock.

(Refer Slide Time: 09:57)

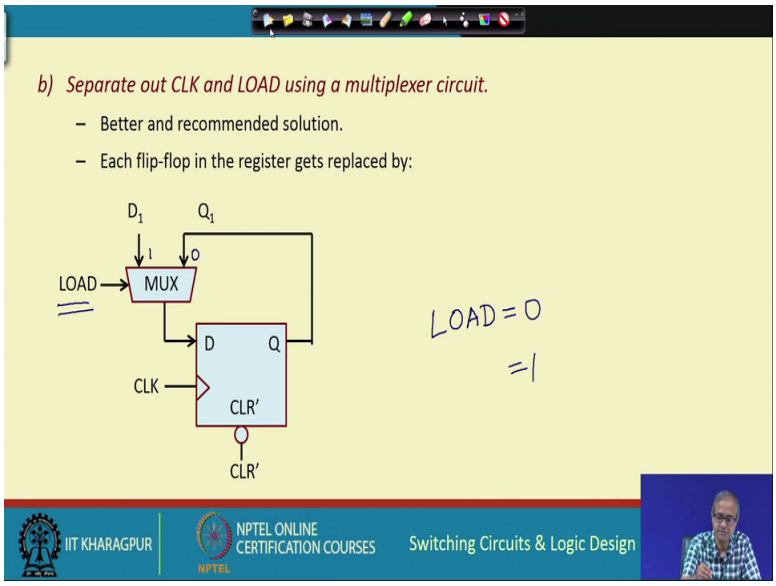


What do you mean by gated clock? We use an AND gate where both clock and load are anded together. So, let the clock come, so the idea is that the clock signal may be coming continuously, but load may be high only for some time. So, whenever both load and clock are high in the output only during that time, the signal will go high the other time it will be 0 and this output of the gate will be fed to all the clock inputs of the flip flop so, that they will be loaded only when required.

Now, the issue is that this method may be very simple, you can simply use an AND gate to implement it. But the problem is whenever let say this load and clock signals are changing at the same time, because you see load is a signal which you are applying from outside. So, accidentally it may happen that you are staring to change the load signal exactly when the clock edge is coming. So, during that time there can be some error in loading or not loading. So, it might get loaded might not get loaded.

So, this solution has a timing issue there can be a timing problem. If you do not apply this signal in exact timing sequence there can be an issue this can cause timing problems, but let us look at another solution, which is better in this respect.

(Refer Slide Time: 11:36)

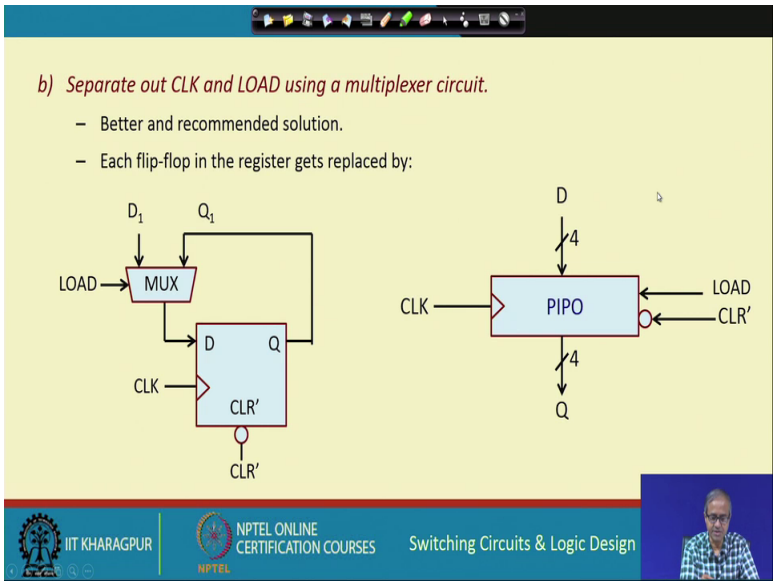


Here we are separating out clock and load we are not combining them using an AND gate like we said, what were doing here we are using a multiplexer kind of a circuit.

The multiplexer will select what will be applied to the D input of the flip flop is it the new data coming from outside or something else ok. So, this approach is the better and recommended solution as a set, where each flip flop in the register will get replaced by a circuit like this just let us try to understand this circuit. Here you have a 2 line into 1 line multiplexer and you have this load control. If load is 0 just try to understand what is happening, if load is 0 which means you are not trying to load.

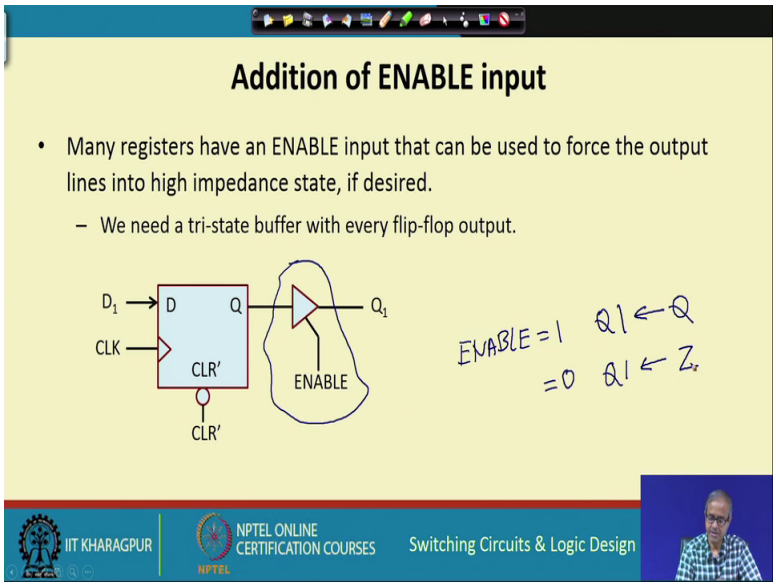
So, whatever is stored in this flip flop should remain. So, in a register there will be a number of such flip flops I am just showing one flip flop, but the clock is coming continuously ok. So, whatever is in D will anyway get loaded with every clock. So, when load is 0 what I do, whatever is getting stored I select that this will be my 0 input of the multiplexer when load is 0, this Q 1 will get selected and come to D. So, the same value will get stored same value will get stored. So, no change, but when load equal to 1 I want to load the external data D 1 which is coming. So, this will be my 1 input of the multiplexer when load is 1 D 1 will coming and D 1 will get stored. So, this is very simple approach this you can use.

(Refer Slide Time: 13:32)



And a register which has this kind of facility, can be represented schematically just very similar to the previous one here with the addition of an extra input called load. So, clock is there clear is there, but in addition there is a load. So, whenever this clock is coming depending on the value of the load either this same value will get loaded or the external value will get loaded.

(Refer Slide Time: 14:02)

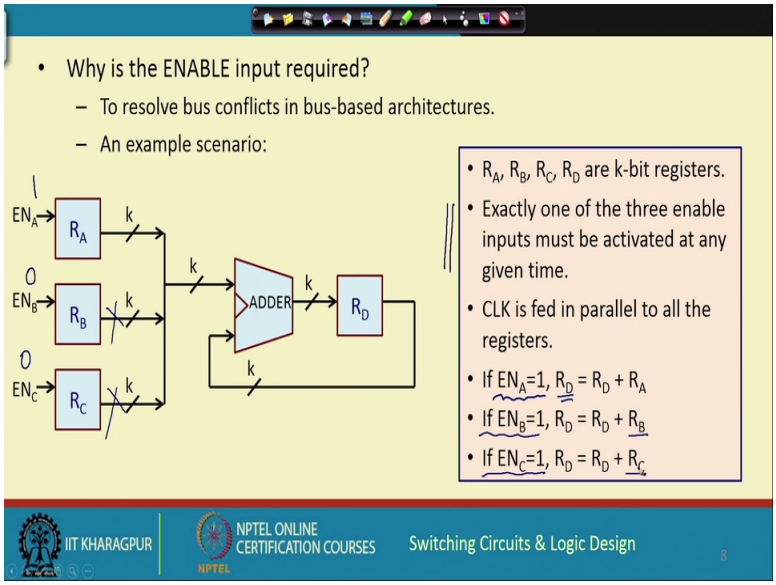


Now, there is another kind of a input which is required in many applications as we shall see very shortly. This is the so called enable input the idea is just follows, I have a

register the register output lines will be giving me some data the data that is stored in register, but there are applications where I want to sometimes electrically isolate the outputs, electrically isolate means I want to send them to high impedance or try states. So, there can be a separate enable input to the register also, where if I enable the register only then the outputs will come, but if I do not enable the outputs will be in the high impedance state this is the idea.

So, as you can see it states that using the enable input, register outputs can be put in the high impedance state, but for this you need some kind of a tri state buffer connected to every flip flop output. So, I am showing this schematic here for one flip flop again. Suppose I have one flip flop of the register the output instead of directly sending to Q 1 here we are using a tri state buffer this is a tri state buffer. Tri state buffer what it does? If you are enable is 1; that means, you are enabling it then the output Q 1 will be getting whatever is there in Q this is the normal mode of operation, but if you are not enabling it enable equal to 0 then this output Q 1 will be in the high impedance state we denote it as Z. So, it is electrically isolated this is the idea behind the enable.

(Refer Slide Time: 16:11)



Now, let us take a concrete example where this kind of enable input is actually required. Now there are numerous examples where you do not have a single register, but many registers. You can take the data from any one of the registers and you want to send it over a common bus to some other place. So, you may need to tie the output of several

registers together, but in a normal circuit you just cannot tie them together right let say if one of the output is 1 other output is 0, if you tie them together there will be a short circuit and a high current may be flowing and the logic level may be either 0 or 1 something in between. So, there can be some error.

So, in a similar situation, when you have a bus based architecture which is very common in computer systems in microprocessors and computer systems, we can use the enable input to resolve bus conflicts. So, I am showing a very simple example scenario; just look at this here this R A, R B, R C and R D these are all k bit registers they are storing k bit numbers k can be anything 8, 16, 32, 64 anything. Now see this 3 registers A B C have separate enable inputs enable A, enable B and enable C and the output of this registers I am tying them together well actually what do mean by tying them together? Let say the first register let say let say k equal to 3 this is R A let say this is R B, I am showing only for R A and R B these are the 3 tying them together means the first output of R A and the first output of R B I am tying together.

Second output of R A and the second output of R B I am tying together and third output of R A and third output of R B I am tying together similarly for R C ok. So, these were k bit registers. So, the output what you get is also a k bit output 3 outputs are coming right. So, you see these registers have k bit outputs, after tying together you have k bit number then I have another register R D and there is an adder. In this adder one of the input is coming from one of this registers and the other input is coming from R D and the output of adder is going to R D.

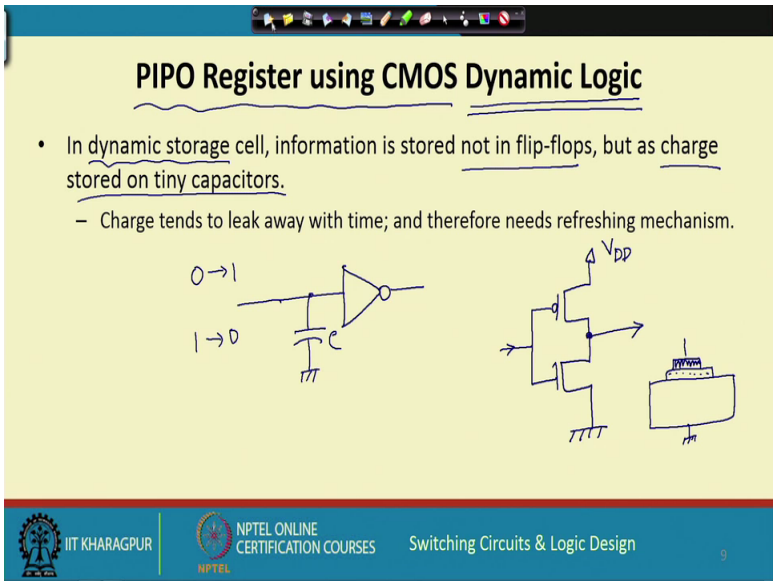
Now, here I am assuming that exactly one of the 3 enable inputs can be active at a time. Exactly one input enable A, enable B, enable C. So, you will be making 1 let say I want to enable R A I make this as 1 I make these two 0 0 exactly one of them will be 1 and clock is being fed to all the registers in parallel. So, what will happen if enable A is active then the output of R A will be coming here, because R A and R B are electrically disconnected they are in the they are in the high impedance state. So, R A will be coming to this adder input and the other input is R D. So, R D will be getting added to R A the result will be stored in R D.

Similarly, if enable B is 1 then R B will get added and if enable C is 1 then R C is getting added. So, here I have shown you a very simple example, but in practical scenario there

can be numerous such examples of data coming from various places and you want to combine them and send them over a single common bus, here this common line that I am showing here this called a bus, 3 registers are feeding that data to a common bus right.

So, this is an example where this enable signal of the register can be very useful ok.

(Refer Slide Time: 20:42)



So, we have seen that how we can implement flip flops using gates. Gates can be implemented using any kind of technology we talked about TTL CMOS and some other technologies as well. Now specifically let us now look at how we can implement registers using CMOS logic because most of the circuits that are designed today are built using CMOS and there are some very simple tricks that you can use in CMOS design to reduce the complexity of the circuits, let us see how.

Now, what we are talking about is here is how we can implement a register using CMOS by using something called dynamic logic. So, what is the idea behind dynamic logic well you all must have heard about the dynamic memories in computer systems. So, the memory that comes to the computer systems are almost universally they are dynamic in nature. Dynamic memory is based on a very similar principal that we want to that we are going to discuss now, dynamic logic

So, in dynamic storage we store information not in flip flops, but as charge stored on tiny capacitors. But where are these capacitors coming from let us take a very specific

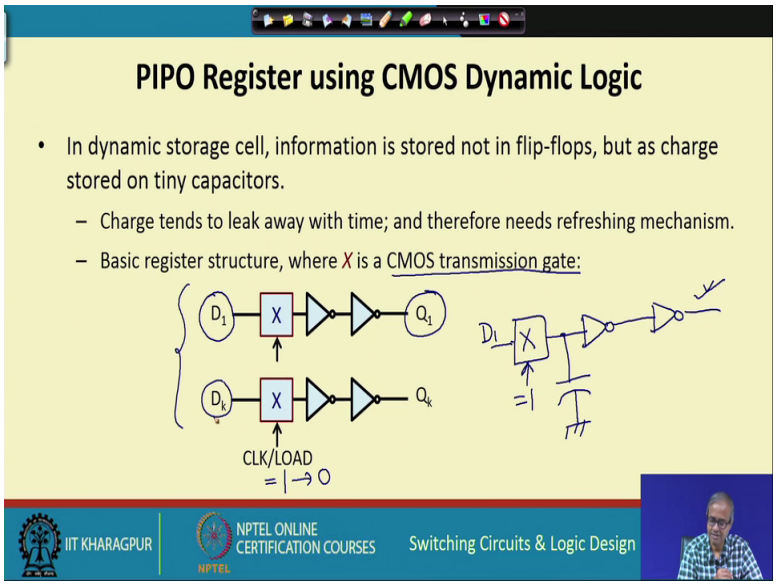
example, let us talk about a NOT gate connected like this. This is a CMOS NOT gate. So, how does a CMOS gate looks like if you recall, there are 2 transistors one is a p type transistor one is a n type transistor they are connected together this is your input and from this common point you take the output, this is the power supply V DD and this is your ground.

Now, if you look at this point, this line is being feeding the gate inputs of 2 transistors. Now for those of you who know how our transistor is fabricated, there is a sub state and the gate is formed on top of the sub state like this and there is a insulating region here, and the sub state is typically connected to ground. So, you see this looks like a parallel plate capacitor there is one terminal this side, one terminal this side and insulating region in between. So, as an equivalence circuit I can say that well as if it is driving a capacitance.

So, when the input I am changing from 0 to 1, as if I am charging this capacitance and when I am changing it from 1 to 0 as if I am discharging the capacitance. Now the idea behind CMOS dynamic logic is that well by either charging or discharging capacitance, I can store 0 or 1 there and you know capacitor as a capacity to store some charge for certain point in time. So, once I charge a capacitor to some required voltage either high or ground representing 1 and 0, then it will retain that charge for certain amount of time. Of course, the charges will slowly tend to leak away, this is not a permanent kind of a storage as you see in a flip flop where as long as power supply is there the data will get stored, but here data will remain for a short duration of time which is of the order of milliseconds.

Now, in terms of the circuit speeds today milliseconds is a very large time ok. So, the charge you are storing on the capacitor can be retained for this period of time after which they will tend to decay or disappear, this is the basic idea behind dynamic storage. So, charge tends to leak away with time and we need to refresh the charge if you want to store it.

(Refer Slide Time: 25:15)

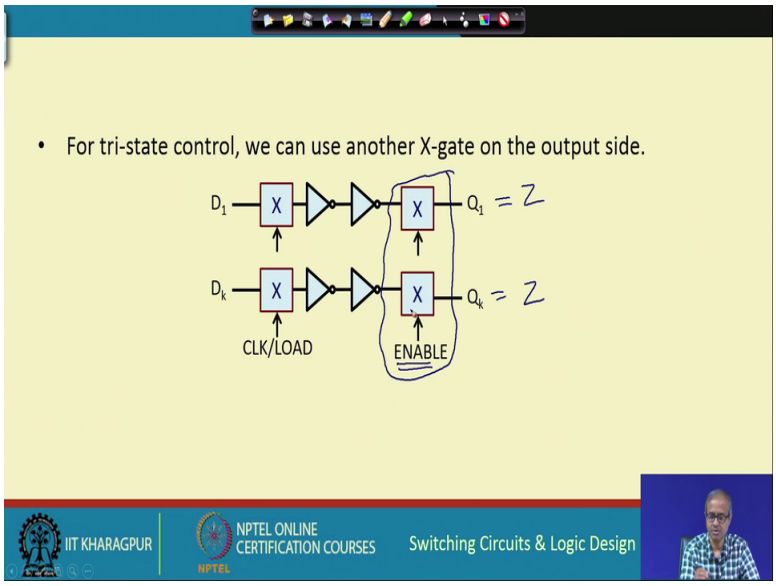


So, let us see how we can implement simple register using dynamic logic. This is register each of this rows represent one flip flop or equivalent to equivalent of one flip flop.

So, you see my data input is coming here my final storage output is coming there, this X is a CMOS transmission gate. So, when clock or load I am setting it to 1. So, this gate is open and you see here after this x there are 2 invertors right connected 1 after the other. So, I am interested in the capacitor here. So, when this gate is open, when this control of this transmission gate I am setting to 1. So, whatever I had applied on D 1 1 1 0 that voltage will get charged on the capacitor and after it again comes back to 0. So, this transmission get again closes. So, this capacitor does not have any path to discharge, but of course, there will be some leakage part very slowly it will discharge, but the data will get stored in the capacitor and there are 2 invertors because whatever you are storing there the same value I want here.

So, 2 inversions will cancel each other ok. So, this is a simple storage. So, if you want a k bit register they will be k such D 1 D 2 to D k right, these are very simple PIPO register using CMOS dynamic logic.

(Refer Slide Time: 27:01)

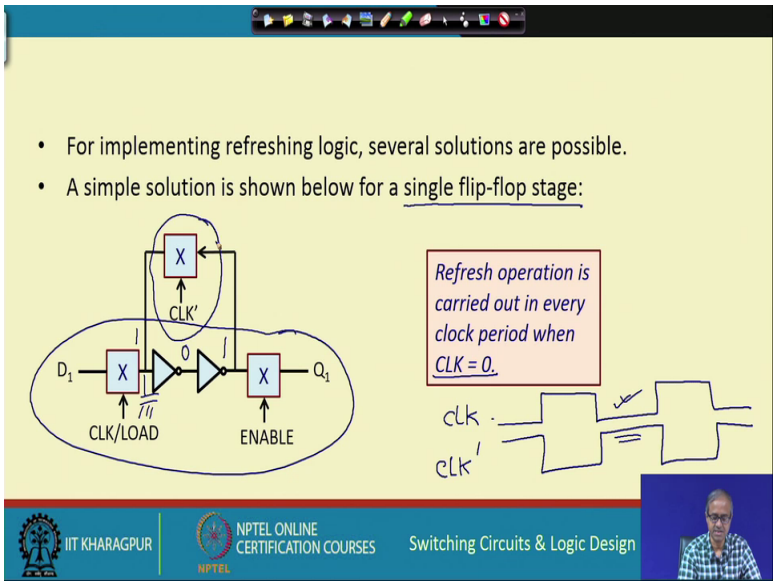


Now, if you want to also add tri state control as we are discussed for normal registers earlier, what you can do we can add another set of transmission gates on the output side and we connect them to another. So, if enable is 1 then this transmission gates will be enabled, and this output of the NOT gate it will come to Q 1 or Q k, but if it is not enabled then outputs will be in the high impedance states simple.

So, you see for a normal convention design, we have to use gates construct flip flops, D flip flop, master slave flip flop so many gates are required, but here you see the design is so simple. In CMOS technology every inverter requires 2 transistors 2 plus 2 4 transmission gates requires 2 transistors and a 2. So, a total of 8 transistors are required to implement register with tri state control a single bit register ok.

But now the question comes that this charges tens to decay or dissipate how do you refresh the charge because we want to store some data over longer periods of time.

(Refer Slide Time: 28:23)



So, there are simple ways available for this, here we are showing one possible solution and you are showing again for a single flip flop stage. So, here we have our flip flop as we have already discussed earlier, there is a clock or load control and there is a tri state enable control, this is something we are adding. There is another transmission gate we are adding here which is being fed with the not of clock, you see clock is coming like this. So, if I say not of clock so it will be just not of this. So, when clock is high this gate is open and whatever there is a D 1 gets stored as charge on this capacitor which is here right.

But when clock is 0 this region, when clock is 0 new date is not getting stored, but this transmission gate is now open because this is activated by clock bar clock bar is high. So, what will happen? Whatever is stored in the output of the second NOT gate that comes back to the input of the first NOT gate and it recharges the capacitor. So, if it was 1, this will be 0, this will also be 1, this 1 is fed back that same value gets charged and if it is 0 that same 0 will be fed back. So, in this way every clock cycle, for every clock period when clock equal to 0, the data is getting automatically refreshed right.

So, by the addition of another 2 transistors you can also have this refresh mechanism. So, with this we come to the end of this lecture, where we have discussed the design of simple parallel in parallel out registers and we talked about different kind of inputs that

may be useful like load and enable, and specifically we looked at how to design a PIPO register using CMOS dynamic logic.

Thank you.