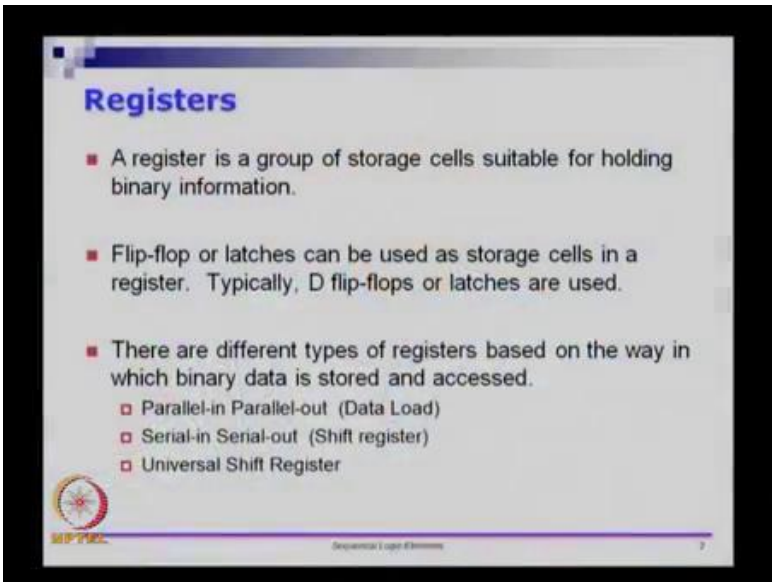
Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module – 22
Sequential Circuits

Hi, welcome to module 22 of week 4. And this is the last module for the week. In this module, we are going to look at the notion of sequential circuits. So, I took a small break from sequential elements and talked about verilog in the last module. And now, we will get back to the sequential circuits. So, what we are going to do is we will be slightly lagging behind in terms of verilog with respect to the class, because I want you to understand the basic concepts first, because before you go and write verilog. So, if I teach verilog and the concepts at the same time, you have to grasp too many things at the same time. So, you will have about a week of material and you will have time to adjust to it and so on. And you will probably even do home works on the material.

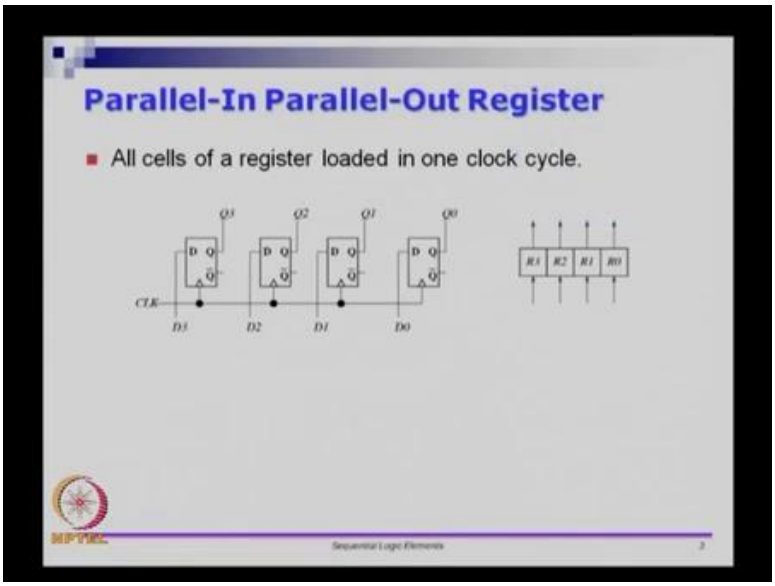
Next week, I will give you some verilog assignment, which is based on the previous week's material. So, that way, these verilog assignments will be one week shifted from the regular whatever I am teaching. This will help you in grasping the ideas first and then going and writing verilog code for it. So, now in this module, we are going to talk about sequential circuits. So, we saw the basic building blocks namely, latches and flip flops. Particularly, we looked at D latches and D flip-flops. So, I am going to use the flip flops and design sequential circuits.

(Refer Slide Time: 01:44)



So, the first thing I would like to talk about is something called a register. A register is just a group of cells. So, it is just a collection of cells. And these cells – these could be either latches or flip-flops. So, usually, we use D latches and D flip-flops, because they are fairly simple to explain, understand. And all it does is storage of data. See if you look at toggle flip-flop; then it is a control condition; when toggle is on, toggle the value and so on. What D does is it just stores whatever is given as input when the clock arrives. So, this is a fairly simple flip-flop to understand and conceptualize. So, I am going to use only D flip-flops in my material here. So, once in a while, I will also talk about D flip-flops. If you go back to your text books, you may also see things like JK flip-flops and so on, which I am not going to address in the course. So, let us take a basic register; register we will treat it as a collection of cells. This collection of cells could be either D latches or D flip-flops. And there are different ways in which you can group the cells together.

(Refer Slide Time: 02:59)



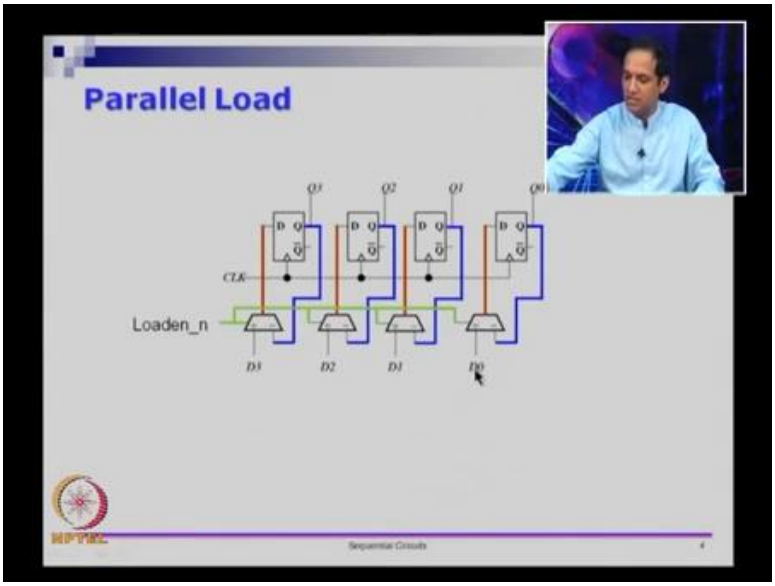
So, namely, you can do what are called parallel-in parallel-out, serial-in serial-out or a universal shift register and so on. We will see each one of them in detail as we go along. So, let us start with the most simplest of the registers. So, usually, the term register is used for something like this. This is called parallel-in parallel-out register. So, what we have is… So, let us look at the picture here. So, we have a set of four cells and these four cells have a common clock. So, you can notice that, there is a common clock. And if you remember the symbol, this is a symbol for a positive edge triggered D flip-flop. So, you have this triangle kind of thing here and there is no bubble. So, it is a flip-flop and it is positive edge triggered. So, it is a positive edge triggered D flip-flop. And there is a single clock signal that is given to all of them. So, all the flip flops are operating in synchrony. And if you see the data input that is coming to each one of them, they are called d 3, d 2, d 1, and d naught; they are feeding four flip-flops. And the output of the four flip flops are q 3, q 2, q 1 and q naught.

So, if I keep all the four data elements ready and when the clock edge arrives, each of these flip-flops will essentially store the current values of d 3, d 2, d 1 and d naught. And this will happen at the same time. So, this is why it is called a parallel-in parallel-out. So, you can see that, all the data inputs are coming in parallel at the same time and all the data outputs – the flip-flop outputs can all be read simultaneously. So, all these bits – q 3, q 2, q 1, q naught can be observed simultaneously. So, this is called a parallel-in parallel-out register. Many times, we will just say that, this is called a register itself. So, usually, we hide all these details and we may use a symbol like this – R 3, R 2, R 1, R naught

with four inputs coming in and four inputs going out. We are hiding a clock specifically. Sometimes we make a notation, which is even simpler than that; which just puts a box and then you will put one arrow in and one arrow out. It says there is one input and there is one output. And this output could be multiple bits and input could be multiple bits. So, if we have four input bits, we will need four output bits though. So, this is called a parallel-in parallel-out register or simply a register. So, one thing this does is it loads whatever value is there in the D inputs; in one clock cycle, they will appear at q 3. So, you place them ready and then you bring the clock edge in; then this data will be ready for the next clock cycle. So, this is a very useful thing; parallel load is a very useful thing.

We will also see something else. So, in this case, if you notice this, one small issue in the circuit. So, if you bring the clock; it takes the data. And if this data keeps changing; then the output of the flip-flops will also keep changing. There is no way in which you can tell the set of flip-flops making this register to hold on to the previous values. So, whenever… So, if there is d 3, d 2, d 1, d naught; if there is any change in it; the next clock cycle, this will appear as q 3, q 2, q 1, q naught. So, it is not letting you. So, it is loading the value; but you have to keep… So, this is assuming that you are loading a value in every cycle. So, sometimes you do not need that; you load a value and then you keep it; and you want to do something with this value for the next cycles and so on. So, if you need something like that; we need something what is called a parallel load register. So, let us see what this is.

(Refer Slide Time: 06:45)



We start with the register that we had earlier. So, we have these four flip-flops, which make the register. So, the D inputs are not shown now. What we are going to provide is you are going to provide a control signal called load enable. This load enable signal – if load enable is 1, we want something to be loaded from the external world; we want to copy something from the external world. If load enable is 0; we want to retain the previous values that we had for Q 3, Q 2, Q 1 and Q naught.

Let us say you want to design a circuit like this. Then let… So, let us see how this can work. So, I am going to put a… So, remember now, I either want to take something from the external world or I want to keep the old value. So, whenever you have a condition like this; you either want this – something to be stored or something else to be stored. Or, in general, if you have if then else kind of condition; then you should think about a mux immediately. So, we have either this or that. So, in this case, if we put a mux here and we are going to have two inputs to the mux; so it is a 2 to 1 mux. And we want to be either take it from the external world or we want to be able to take it from the previous value that the flip-flop was storing. So, let us do it for one flip-flop first.

So, if you now look at this picture; either the external world input would come into d or the previous value that is stored in that flip-flop will be fed back based on the condition that you put for the select signal here. So, I hope that is clear. So, based on what you put in select; if you put select as 0; then it will load from the external world. If you put select as 1, you would get it from the previous value itself. So, note that, the D flip-flop internally has some feedback path; and outside that, we also have a feedback path going from the output of the flip-flop to the input of the flip-flop. So, this is a combinational feedback loop, which starts from Q goes through mux and goes back to D. So, this is a combinational loop. Now, we can…

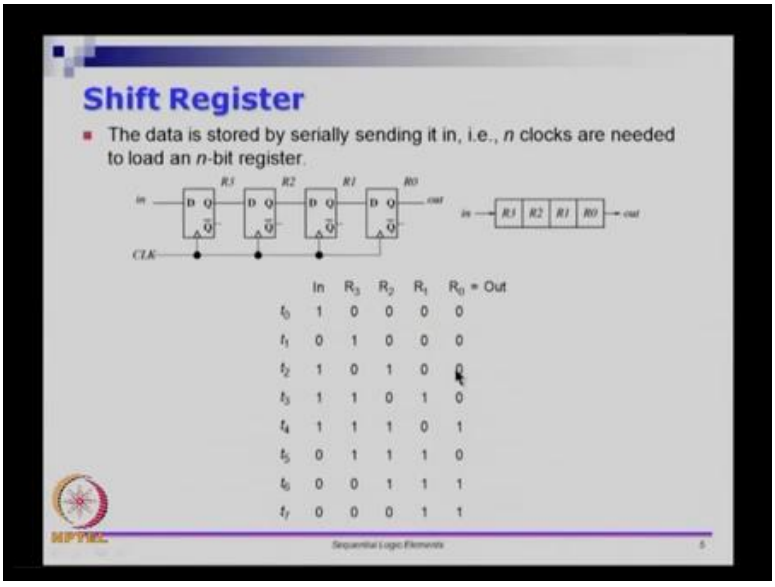
So, wanted a load enable signal. So, instead of load enable, I am going to have load enable bar. So, if load is 1; load enable bar is 0. So, what we want is when load is 1; you want to take it from the external world; however, I am putting load enable bar. So, what this will do is if load enable is 1; load enable bar is 0; which means you take it from the external world. If load is 0; you want to keep the old value. So, if load is 0; load enable bar will be 1 and you can see that the mux is choosing the input on pin 1. So, that is coming from Q So, this takes care of either retaining the old value of the flip-flop itself or being enable to load a new value. So, if you do not want to load a new value, you then make load enable equal to 0 or you drive 1 to this pin – to this select pin. So, it will keep

the old value that is stored in this flip-flop.

Now, I can do the same thing for the other flip-flops also. So, I did it for Q 2 flip-flop and for all of these flip-flops. So, all of them have a signal. So, they have D 3, D 2, D 1, D naught coming into the pin 0 of the multiplexes. Pin 1 has the respective outputs of the flip-flops. We still need something which connects all the load signals together. So, let us put that in. We can see that in the green line here. So, the green line is a load enable bar, which is connected to the select lines of all of them. So, what we have is we have a mechanism by which if we make load equal to 1 or essentially load enable bar equal to 0; then you will read it from the external world in one clock cycle.

So, in one clock cycle, each of the flip-flop will be able to take the new value and give it to say D input. However, if load enable is 0; load enable bar will be 1; you will take the same value and loop it back; you are going to keep the old value as it is. So, this is what you have in a parallel load setup. So, it is a fairly interesting setup and it gives you one clock cycle access to the data. And the key thing is Q 3, Q 2, Q 1, Q naught are still retaining the capability of being red in one clock cycle. So, I can read all of them in one go. So, D 3, D 2; D 1, D naught is plugged in these pins and we are done.

(Refer Slide Time: 11:39)



Now, let us look at another structure, which is called the shift register. So, the shift register is actually… So, the previous one, we saw was parallel input parallel output. In this one, what we have is something called serial input serial output. So, let us look at the picture. We have as before, we have four flip-flops and we have a single clock going into

all of those. So, it is going to be a synchronous circuit. All of them are aligned to the same clock. So, the key difference is we take the… So, there is only one input bit. We do not have multiple bits; we have only one input bit. We take that input bit and give it to one flip-flop. The queue of that is connected to D of another flip-flop and so on.

So, I can think of it as people holding hands left and right. So, from one side, you get the input and one side you give the output; it is like that. So, what you have is you have an input bit that is coming into D; then you have Q; the Q of that is going in as D for the R 2 and so on. So, pictorially, if you want to represent it; it is usually represented in this form. So, you have a single bit coming in from the left-hand side and a single bit that will go out to the right-hand side. And we call this; this is shift register, because it shifts bits one at a time.

So, let us assume for a while that, R 3, R 2 and R 1 and R naught are all zeros to begin with. Let us assume that, all these flip-flops somehow had 0 to begin with. And I am going to shift in a sequence of bits – one bit every clock cycles. So, I am going to shift in 1 in the zero-th clock cycle; 0 at t 1; 1 at t 2 and so on. So, this is some input that is coming from the external world. And the R naught is actually tabbed out as output. So, this is going to the external world. So, we have something coming in and we have something going out. There is only one input bit and one output bit.

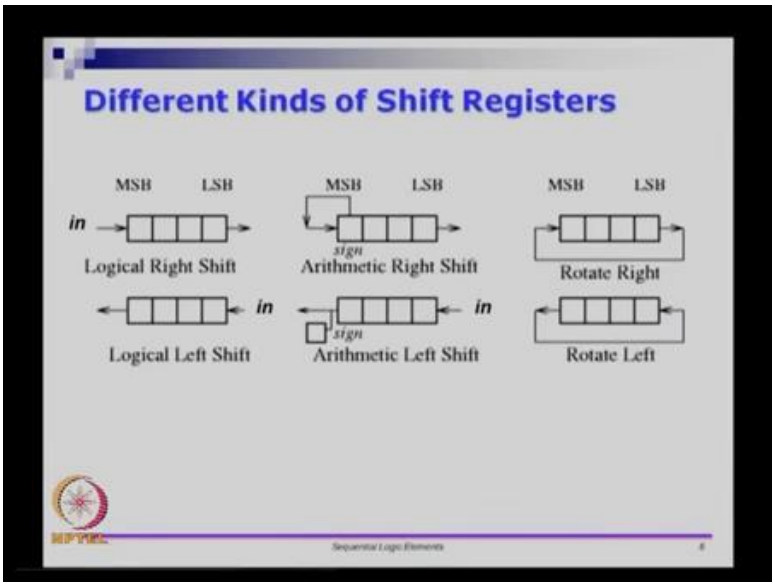
So, let us see what this does. So, if you put t naught equal to 1 at some time point. So, you will see that, this input will appear in its output one clock cycle later. So, this one shifts to this location at time t 1. So, the input was made available in the previous clock cycle to this flip-flop R 3. It comes out as output of R 3 at t 1. And if you look at these three 0's; so this 0 gets shifted here; this 0 gets shifted here; and this 0 is going as output. So, R 1 had a 0. So, if you if you look at it R 1; this input is a 0; and D input of R 0 is 0. So, when it gets out, it will come out as 0. So, you can think of it as arrows going diagonally below. So, this data input is coming as data output here; this data input is coming as data output here; and so on. So, that is for t 1.

Let us say a t 2… At t 1, I gave 0 as the input to this line. So, t 1 is 0 at this input. So, earlier R 3 had 1; but because it is shifting in; this 0 gets shifted in; this one which was input of R 3. So, input of this flip-flop goes as the output of that flip-flop now. So, this flip-flop gets one as this output; this 0 gets shifted here; this 0 gets shifted here. So, what you will see is the output is coming out of R naught and input is coming into R 3; and

there is one bit that is shifting to the right in every clock cycle. So, let us take this time instance for example. So, at this time instance, R 3 is 1, R 2 is 0, R 1 is 1 and R naught is 0. And we are shifting in a 1. So, this 1 gets here; this 1 gets here; this 0 gets here; this 1 gets here; and 0 get shifted out.

So, whatever is coming out of R 0 is shifted out. If you notice, there is 1 0 1 1 here and you will see that there is 1 0 1 1 here. So, whatever came as input is coming out as output; only that… So, t naught was given at zero-th clock cycle and t 4 will come out at fourth clock cycle. At t 4, you get 1. At t 1, we gave input of 0. At t 5, we will have 0 out. That is happening because there are four flip-flops in the sequence here. If we had five flip-flops; then instead of t naught coming out at t 4 to the external world; whatever we gave at t naught would come at t 5. So, this is the picture that represents the shift register. So, we have one input bit and one output bit. I hope that this is clear to you. And what is happening in this case is it is shifting all the bits to the right. So, we are starting from this side to this side. So, we will call this flip-flop f 3 and this is f naught. So, it is shifting bits from left to right. This is called a right-shift register to be specific.

(Refer Slide Time: 17:00)



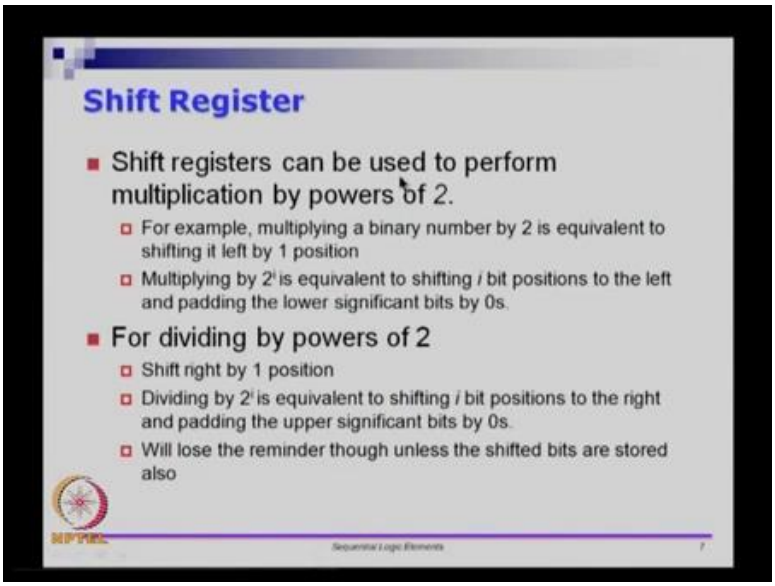
There are other kinds of shift registers. So, let us look at right shift. In right shift, the D of this MSB bit in our previous example was Q 3. The Q of the f 3 flip-flop was connected to D of the f 2 flip-flop and so on. So, in that case, we get a logical right shift circuit. You can also connect them the other way round. You take the Q of D naught connected to D of flip-flop 1; take Q of that connected to D of flip-flop 2 and so on. In that case, you have to go from right to left. So, that is called a logical left-shift register.

You can also do a few other things. For example, you can take the output that is coming from one side and give it as input to the other side.

So, let us say this is a right-shift register; but I take the output coming from the right-shift register; give it as input to the right-shift register. This will just rotate the bits one cycle at a time. So, if it starts… So, let us say I put symbols instead now; I put abcd instead of bits; if I put abcd; then d will come here; it will become dabc. Then another cycle c would have come here. So, it would have cdab and so on. So, this kind of thing you might have seen in rolling displays in theaters and other places. Things keep just rolling from left to right or right to left; you might have seen some of these things. So, this is equivalent to that; only that we are operating one bit at a time and it is continuously rotating.

So, sometimes, you also see things, which go from left to right or right to left and they do not come back. They are equivalent to the shift registers. You might have seen displays like that also. Then there is this specific thing. We will get to this later; but in this picture, what happens is you are having right shift. So, this is shifting right except that the more significant bit is shifted to itself. So, this is called arithmetic right shift. And similarly, if you have a left shift and if you take this bit here and copy it somewhere; that is called an arithmetic left shift. We will look at these two things when we look at signed numbers later. As of now, let us ignore these two pictures; but all of these are essentially a sequence of flip-flops that are all arranged in such a way that output from one of the flip-flops is going to input of the other flip-flop and so on; which direction it gets connected and what not; tells you what kind of shift register it is.

(Refer Slide Time: 19:27)



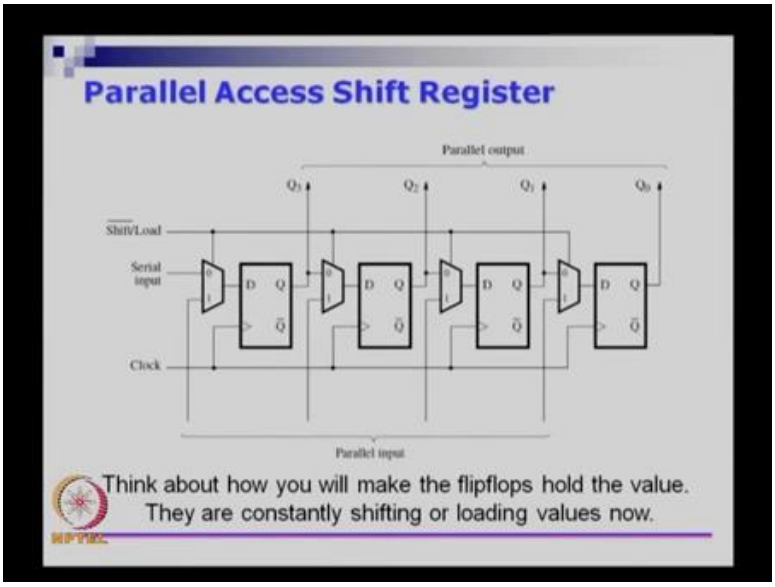
So, shift registers are quite useful; they can be used to perform multiplication by powers of 2. So, for example, multiplying a binary number by 2 is equivalent to shifting it left by one position and adding a 0. So, if I want let us say number 3, which is 1 1 in binary; if I want it to be multiply it by 2; all I have to do is take and shift it by one position to the left. So, if I have 1 1, shift it by one position to the left and put a 0 there – 1 1 0. So, 1 1 0 is number 6. You can extend this idea if I want to multiply by some power of 2. So, let us say I want to multiply by 32, which is 2 power 5; I do not have to really multiply the numbers; instead I can take the number; shift it by five positions to the left and put zeros in the last five positions. That will give you the original number into 32.

So, just go and work it out on paper; you will see that, if you take a number shifted by i bits and put 0's; now, it becomes whatever the previous value was into 2 power i. Similarly, you can do division; you can take a value and then right shift by one position, right shift by one position is equivalent to dividing by 2. And if you right shift by i positions; then it is equivalent to dividing by 2 power i. The only thing is when you divide, you get the quotient; you do not get the reminder. So, the i bits that gets shifted out when you shift right or actually forming the reminder when you divide by 2 power i. So, this is something that you have to store if you want the reminder also; you will keep the…

So, if I put a value let say n in a 5-bit register. So, let us… I put some value n in a 5-bit register; if I right shift it two times; the equivalent value will be n divided by 4 or the lower… So, you know the seal and floor functions; we will give you floor of n divided

by 4; which means the integer that is lesser than n by 4 and the closest to it. So, this is very useful because if you are going to do arithmetic operations, which we will come to later; you do not… And if it is involving multiplication by 2 or multiplication by any powers of 2; you do not have to use what is called a multiplier; you can actually put it in a register and use only shift operations and do the multiplication. Similarly, if you are going to divide by some 2 power something; you do not have to use a divider; instead you can just right shift by i positions. So, this is a very useful thing to know. We will see this any way in a little detail later.

(Refer Slide Time: 22:16)



Then, there is also something called a parallel access shift register. So, let us see this circuit now. So, all these circuits are minor variations of what we have seen earlier. So, it seems to have a mux. So, do not look at the mux and immediately say that, this was the parallel load that we had earlier. Look at all the connections, that is there and decide what it is actually doing. Let us see this one here. So, it seems to take the same clock input to everything; so which means it is a synchronous circuit. And you are seeing that, there are four lines that are coming out and there are four lines that are coming in. So, it seems to have something, which is coming in in parallel and there is also something that is going out in parallel. So, this part of it resembles just a register. However, if you notice, there is some mux, which is connecting various things. So, let us see what this is actually doing.

So, if you go and look at the mux here; there is a select line that is coming in, which says either shift bar or load. So, the meaning you can attach to this line is you can call it either

shift bar or you can call it load. So, let us assume that, for a while, this line is a 1. If it is a 1; let us see what this circuit does. If it is a 1; then this mux is going to select whatever is on the pin 1 of each of these muxes. So, this pin will select what is on pin 1; this mux will select what is on pin 1 and so on. And if these are external inputs D 3, D 2, D 1, D naught; they are the ones that are going to appear in the D input. So, D 3, D 2, D 1, D naught – if you put here; they will go through the mux, because you put load equal to 1 and appear as D inputs for each of these flip-flops. So, you are able to get something like a parallel load.

Now, if load is 0 or shift is 1; so the meaning you attach is either shift bar; you either call it shift bar or you call it load. So, if this line is 0; it means shift bar is 0 or shift is 1. If shift is 1, then shift bar is 0. Let us see what is in the 0 pin of all the freezed multiplexes. 0 pin is for of this mux is taking a serial input; it is taking one bit input; it is placing it at D. And if you look at the 0 pin of this mux; it is taking Q 3 and giving it as input to flip-flop 2. Then Q 2 is given as input to flip-flop 1 and Q 1 is given as input to flip-flop 0. And the output of flip-flop 0 is going out of Q 0.

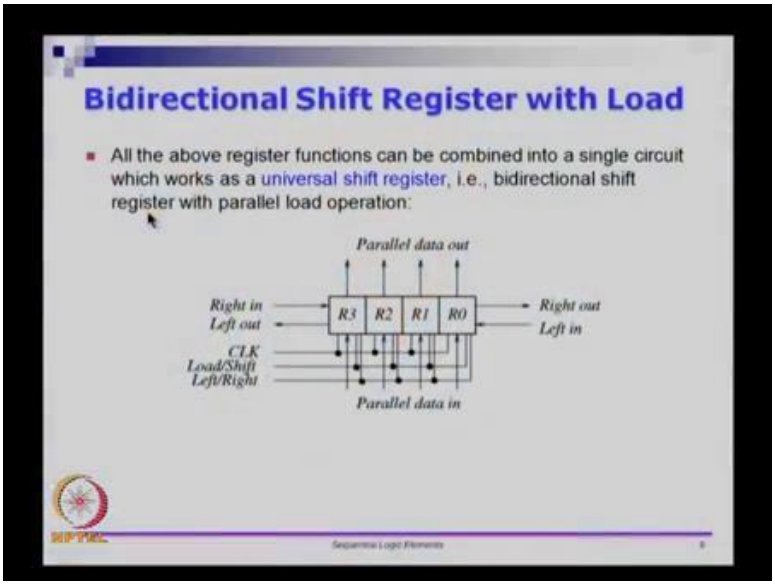
So, if you look at this part of the circuit; it resembles what we had for a shift register to be more specific; it is a right-shift register. So, what we have is we have put a mux in such a way that, it either adds such a shift register, which is shifting right or it can do something in parallel. So, the key thing is shift register requires n cycles to read the value of a register. So, if I want to see what are the contents of these four flip-flops and if I can operate only in the shifting mode; then I need four cycles to see what these bits are. So, this bit I can see in the first cycle; this bit in the second cycle; this one in the third; and this one in the fourth. So, shift registers need more cycles for you to see what are the actual bits that are stored currently.

And similarly, if you want to put a value into the register using only serial input; then it will need n cycles also. So, loading the register serially takes n cycles; getting the values, which the register has also takes n cycles. However, loading in parallel takes one cycle; reading in parallel takes one cycle. So, a combination of these two is sometimes useful because sometimes you want to load in parallel. And let us say I want to shift by… I want to divide a number by 8; then I will I can load the number in parallel in one cycle; then I will shift right three cycles and I will load this value out in one cycle. So, you can do something like that if you have a combination of serial and parallel input and serial and parallel output. If you do not have that, you will burn yourself in either loading it or

in unloading. So, for this specific problem that I gave you; if I want to divide a number by 8; then parallel load can bring the value in one cycle, but you cannot do anything with it. Whereas, if you do it in the sequential manner; you will need four cycles to load the 4-bit value; then you will need three more cycles to divide this and then you can get the output. So, that is cumbersome. So, parallel input and output is useful if you want to quickly read the values and if you want to do any operations like division or multiplication and so on. For that, you need shift. And this is a circuit that actually gives you both.

So, I am going to give you a small piece of home work here. So, one thing this is doing is it is either continuously loading or it is continuously shifting; it has to do one of those, because shift bar slash load will either be 0 or it will be 1. If we keep doing this right; you are either continuously loading or you are continuously shifting. What if we want to keep the value? I do not want to shift; I do not want to load; I want to just keep the value; which means the queue that you have from here should also somehow go back to the mux. But, I have only a 2 to 1 mux. So, see what change you will make to the mux, so that you can either shift in or you can have a parallel load or you can retain the old value. So, come up with a circuit, which does that and see how you would control it and so on. So, you want to be able to keep the value, shift the value in from let us say left to right and you want to do parallel load; think about that.

(Refer Slide Time: 28:23)



So, designers also design something called bidirectional shift register with load. This is like a universal shift register. So, let us see what it does. So, you have a register. In this

case, it is a 4-bit register: R 3, R 2, R 1, R naught; it seems to have a right input as well as a left input; it seems to have a right output as well as a left output. And then it has a parallel load coming in from the bottom and a parallel load going to the output. So, it can behave either as a left shift circuit or as a right shift circuit or as a parallel output that you can get; or, it may just load; it may just retain the value. So, something like that is called a universal register. In a universal shift register, you can either shift in from the left or shift in from the right or you can load in parallel or you could actually just keep the old values. So, the D input for each of these flip-flops will either have something from the left side or the right side or the data input from the external world or its own previous value. So, go and think about how you will design a circuit for this. So, the whole circuit is not shown; you are only seeing boxes here.

Go and see how you can design a circuit which can do this. So, this is another homework that I want to give you. So, how do you implement universal shift register using D flip-flops and basic logic gates? So, what we have done so far is we have seen various things leading to this notion of a register. So, in this week, we did quite a few things; we started with the basics of flip-flops. So, we started with latches; then we did flip-flops; then we looked at how to connect the flip-flops together in different forms and so on. So, the most meaningful connections were given here. Either you can do things in parallel or you can do things in series; you can connect them in series or you can connect them in parallel. So, these are meaningful connections of a group of flip-flops. That is why we have a name called register for them. And in the last picture, I show here; it is called a universal shift register. So, it is a very useful thing. This on the same input, you can divide, you can multiply by powers of 2, you can load them or you can just keep the values. So, something like this is actually there in processes. If you have done assembly programming, the operations like left shift, right shift, move and so on are all possible only because there are registers like this inside your microprocessors. So, this brings me to the end of week 4.

And we have already done quite a bit. So, we started from basic Boolean algebra. And in one month, we have seen various things leading to shift registers right now. So, the next set of videos whatever we have from now on will ramp up the style and contents of the course even more. And we have seen bits and pieces of verilog code; I would like you to experiment with verilog code as much as possible. Week 4 will also have some contents with verilog. So, I suggest that you go and do, practice problems in verilog. And we will

see; as we go along, how to use these, how to write various things in verilog. So, as of now, you are writing basic combination circuits maybe. Slowly, we will start writing code for sequential circuits and so on. So, that brings me to the end of week 4. Thank you very much and I am hoping that you are enjoying the course. So, if you have any issues about the course or anything, please e-mail me on the forum. So, I actively check the forum. So, please reply back on the forum and give any constructive or any feedback – positive or negative about the course so far. We are done with four weeks now; anything that you have so far, please go on the forum and start threads either constructive or anything. Even if you have some negative comments about anything; please put that on the forum, so that we can correct it right away.

Thank you very much and see you next week.