Digital Circuits and Systems
Prof. Dr. S. Srinivasan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Lecture #21
Shift Registers

(Refer Slide Time: 2:00)

Digital Circuits and Systems

Summary

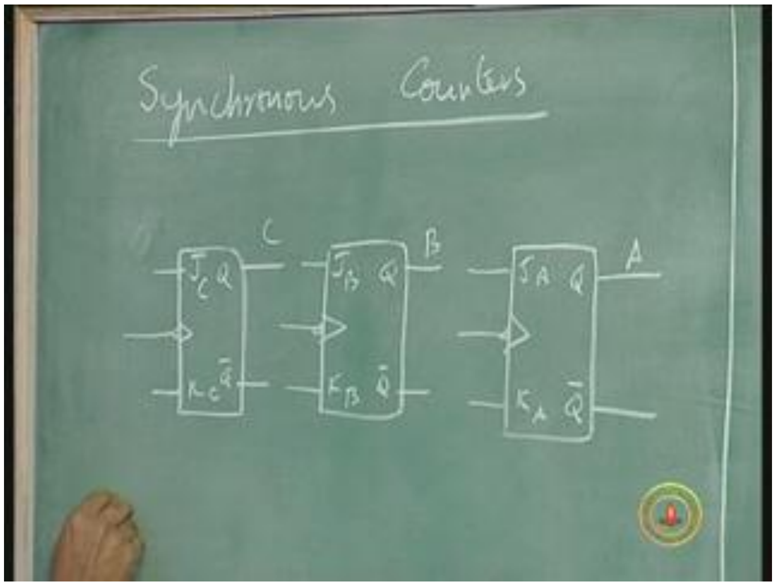
* Synchronous Counters and Registers
* Shift Registers and its Types
* Universal Shift Registers

Lecture #21 Prof. S. Srinivasan Department of Electrical Engineering

We were discussing counters, use of flip-flops in counter design, up count and down count or counting from an arbitrary count to another arbitrary count or down from an arbitrary count to another arbitrary count. In all these cases we use what is known as a ripple configuration that is the output of a flip-flop drove the input of the next one, it was used as a clock for the next flip-flop. So in that we saw that the delay of the flip-flops play a role in how fast you can count. That's not a very desirable thing because as the number of flip-flops increases or as the total count that you want to count increases the propagation delay of the repel counter increases linearly the number of flip-flops so there comes a point where the total delay of the time it takes for a count to stabilize to the next count is longer than the clock period. When that is the case then you can never have a stable count.

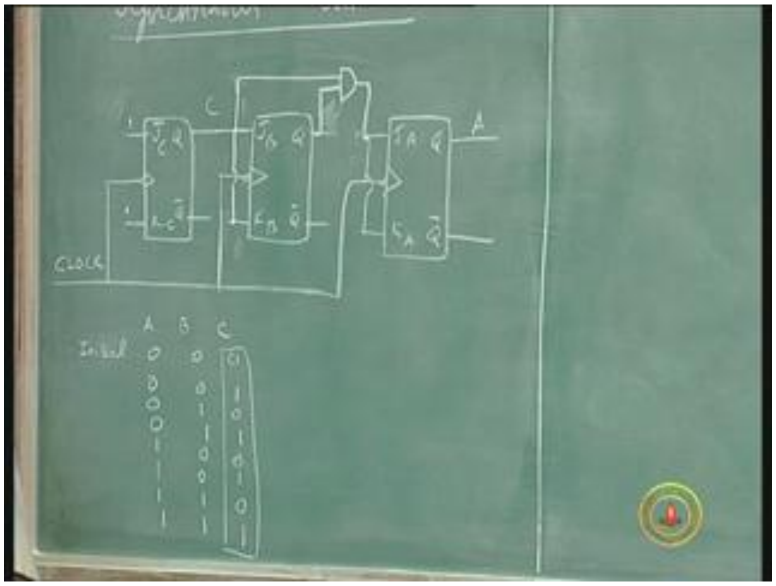
We sort of wondered whether it is possible to have a counter which will be independent of the number of stages we need to count. That can happen only if the clock pulse is applied to all the flip-flops uniformly at the same time. Because any change in the output based on the input of the flip-flop can be triggered only after the clock goes high. So when the clock has to be active in order for the flip-flops to settle down in the output you need to supply the clock and if you do not want the number of stages of the flip-flops to decide the counting period the delay you need to apply the clock pulse to all the flip-flops at the same time. That means you are feeding the clock to the first flip-flop we have to feed the system clock the input clock to all the flip-flops at the same time. Then how do you control the currents, because once you put a j equal to one the k is equal to one you give a clock all flip-flops toggle. That means if all the flip-flops are 0s to start with they will become 1s after the next clock pulse so the flip-flops will go changing from 0 0 0 0 to 1 1 1 1 and so forth. We don't want that, we want to count in a binary sequence. So what we have to do is to look at the circuit of a counter we call this synchronous counters. The synchronous counter is the counter in which the clock is applied to all the flip-flops. So let us take a 3-bit to start with easy to handle in a classroom. I can have the same negative edge triggered or master slave configuration as we had earlier I will call this a b c this will be A, this will be B and this is C, Q Q bar so this will be A, B and C.

(Refer Slide Time: 6:30)



I want to apply the clock input to all the flip-flops so we put a 1 1 here let me write the counts. So let us say to start with we want to have 0 0 0 initial during the first clock period let us say after the first clock pulse. Let me now write the sequence I want, I want this sequence (Refer Slide Time: 7:49) this is my desired sequence natural binary sequence look at the flip flip-flop C this one; 0 1 0 1 0 1 0 1 so it needs to toggle at every clock pulse so there is no problem with this JK flip-flop because once you put J is equal to 1 and K is equal to 1 and give the clock pulse it is going to toggle. So I will not have any problem with this J is equal to 1 K is equal to 1 and I will get the desired sequence at the flip-flop C output. Here of course if I put JK 1 1 for the second flip-flop and B if I put 1 1 here this will also change so it will be 0 1 0 1 0 1 because I am giving the clock to all the flip-flops and I put a 1 1. So whenever I put a 1 1 in this toggle mode and as in the toggle mode when you give a clock the output compliments itself so from 0 0 it becomes 1 1 then 0 0 1 1 which I don't want but I want 0 0 0 1 1 0 1 1.

(Refer Slide Time: 9:45)



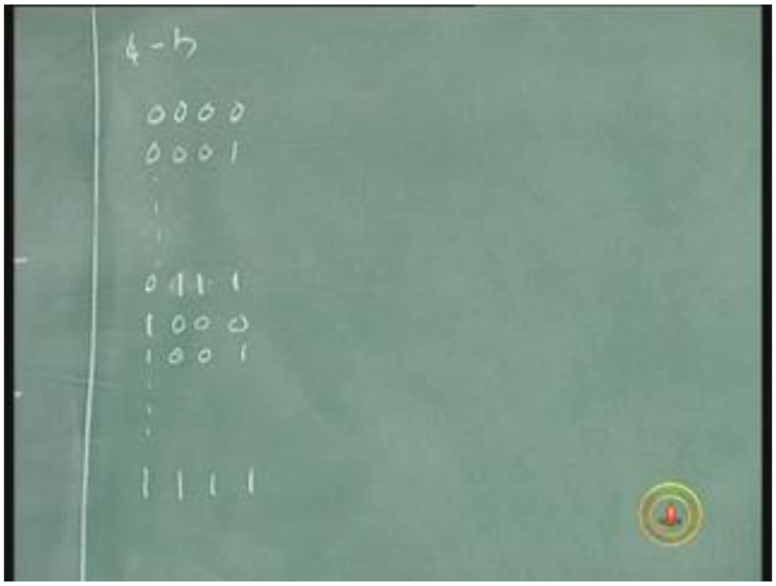
So instead of doing this what I will do is I can use this information that is when this becomes 1 when C is 1 at the next clock pulse B has to become 1. So instead of tying this C, J and K unconditionally to 1 1 we will connect it like this. Suppose I feed the output of c into the input of JB the second flip-flop input, and first this was 0 but even though the clock was there when this change state this would not change state because the output of c was 0 and that 0 was given here and a 0 and 0 so we will also put this character JK here both J and K I will give one value 0 0 so during the second pulse when this became 1 from 0 this does not become 1 from 0 because J and K are both 0 0 so 0 0 of the clock pulse applied remains same in the JK flip-flop.

Now the next pulse appears after 0 0 1 by now JB and KB has been kept to 1 in the meanwhile after this pulse so when the next clock arrives at that time there will be a toggle mode here there will be a toggling operation in this flip-flop (Refer Slide Time: 10:55). This will toggle anyways because I have connected with JK 1 1 so it toggles with every clock pulse, this will toggle after this because J and K are 1. In the next clock pulse it will not toggle because this is 0 so for every other clock pulse only the J and K of the second flip-flop becomes 1 for every second pulse, for every alternate pulse J and K of the second flip-flop become 1 1 so the second flip-flop will toggle only once in two clock cycles.

I am going to repeat this argument now. Third flip-flop I want to toggle only once in four clock cycles, I want this to be connected to 1 and remain one for every four clock periods, for four clock periods J and K of the third flip-flop will remain 0 and at that time there will be no change in the output for the third flip-flop the A flip-flop and for four clock pulses it remains 0 and for another four clock pulses it becomes 1. Now, when do you want the transition to happen? At the clock pulse after the count 1 1 was reached here. after the 1 1 was reached here I want this to change once but that as to remain again four times for four clock periods. So what I am going to do is to take these two into an

AND gate. So when both B and C are 1 1 at the fourth clock pulse immediately thereafter JA and KA becomes one but the clock pulses have gone. We can only do it next time. So after that it will now become 1 1 and when it is 1 1 at the next clock pulse this will toggle. I can keep this going with this fourth position for 4-bit counter.

(Refer Slide Time: 14:06)



It is a 3-bit counter only 3-bits three positions are there. And here these three becoming 1 1 1 will make the fourth flip-flop toggle and remain the same because after that not all of them can become 1 at the same time. Again all of them will become 1 at the same time here then it will become 0 which is same as this so it will repeat. So when all the previous flip-flops have changed into 1 you want the toggling of the next flip-flop and that has remained for several clock cycles before the next change can happen. This is the concept of a synchronous counter wherein the clock is given to all the flip-flops and the change in the queue controlled by the outputs of the flip-flops.

Earlier we tied J and K of all the flip-flops to 1 so every flip-flop toggled but then clock was delayed, the first flip-flop received the input clock and the system clock as they call it and the second flip-flop received a clock pulse which is half the frequency of the first, the third flip-flop received a clock pulse which was one fourth the frequency of the first and so forth. Therefore we divided the clock and by using the lower and lower frequency clocks we are able to get the toggling action slower and slower. The idea is that the first flip-flop has to toggle very very fast, the second flip-flop to toggle at a relatively slower rate this slower compared to this and so forth. Toggling is required but need to be controlled. Earlier we controlled it by means of a clock but now we are controlling it by means of the output of the flip-flops.

What is the advantage of this over the other one, synchronous and ripple counter? Repel counter by the way is also called a non synchronous counter asynchronous something which is not synchronous is asynchronous. So asynchronous counters or ripple counters are the ones where we use the clock of any given stage from the previous output.

In the synchronous counter clocks are supplied to all of the flip-flops at the same time, the control of the toggling action is done by the outputs of the previous flip-flops. The advantage is going to take place only after the clock pulse has arrived because this would have settled down. For example, take the transition from this to this, as soon as this clock has arrived this would have settled down and for the entire duration of the clock this is going to remain as 1 1 so my J and K would have been fixed as 1 1 locked before my clock is due by this flip-flop, J and K will become 1 1 long before the clock comes.

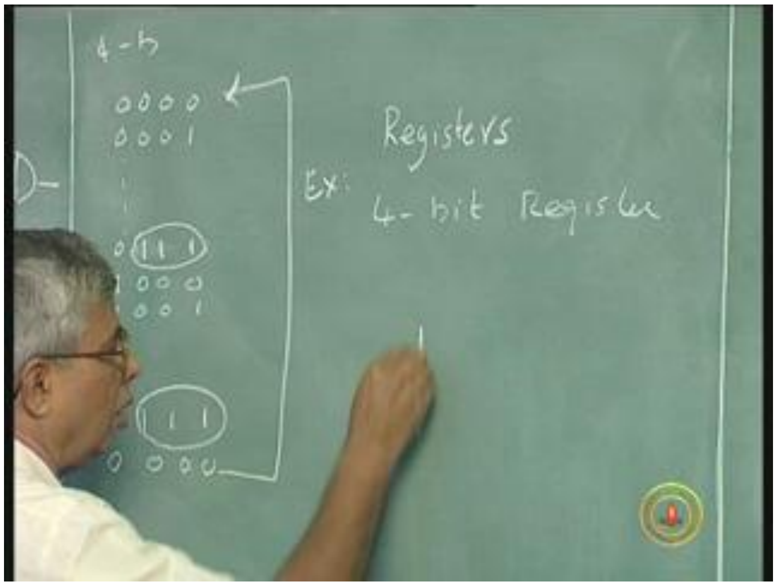
When the clock comes all it needs to take is the clock period delay so the delay of the flip-flop. So any flip-flop will change the output after one propagation delay. Earlier the second flip-flop will change the output after two propagation delays, third flip-flop will change the output after three propagation delays, fourth flip-flop will change the output after four propagation delays and so forth. That is why we found that it is not possible to carry on as a long chain because the total propagation delay should be less than the clock period. That condition is not there anymore. We have to make sure that the delay of each individual flip-flop is less than the clock period which is adequate. It can be amply supported or taken care of it should not be a problem.

All systems use flip-flops and counting mode and most of them use this synchronous mode of counting. The only disadvantage is you need extra hardware AND gates. Actually you don't need to have three input AND gate because I have the output of this I have to take the output of this and put it in this so every time I put two input AND gate I take the output from the previous AND gate as 1. So if you want to continue this it will be like this. I don't have to go for three input AND gate or four input AND gate, two input AND gates will do for each stage. So we stop with the flip-flops the synchronous counters non synchronous counters asynchronous counters ripple counters.

Basically flip-flops are used for counting starting at any counter and terminating at another counter all within that 2 power n minus 1 range. For n flip-flops we cannot have more than 2 power n states. So if I want to start from 0 it has to be 2 power n minus 1 within that I can start at any count and end at any counters either up or down and they can be synchronous or asynchronous all that we have already seen. So what is the major use of flip-flops other than counting? Counting is one thing of course. What is the original intention of starting sequential circuits here? I said we need some storage element to store a bit of information; the latch was a storage element. So now the main use of the flip-flops is in the design memory. Even though we talk about one bit flip-flop a flip-flop which can store one bit in practice we never have words of one bit.

Usually a word is 4-bits 8-bits sixteen bits thirty-two bits like that. for example, if you take a code ASCII code I said translate your alphabets and numerals into binary code such a code is called ASCII code I said we have seven bits for that and we may have an extra bit for parity so 8-bit code. So I want to store that code corresponding to a particular alphabet. I need 8-bit eight flip-flops. So instead of building memories using single flip-flops we can also have circuits or devices sold in the market which can take several bits at the same time and store it so such a combination of flip-flops is called register. You may have seen this word being used in several digital books. Register is nothing but a bunch of flip-flops a group of flip-flops.

(Refer Slide Time: 22:05)



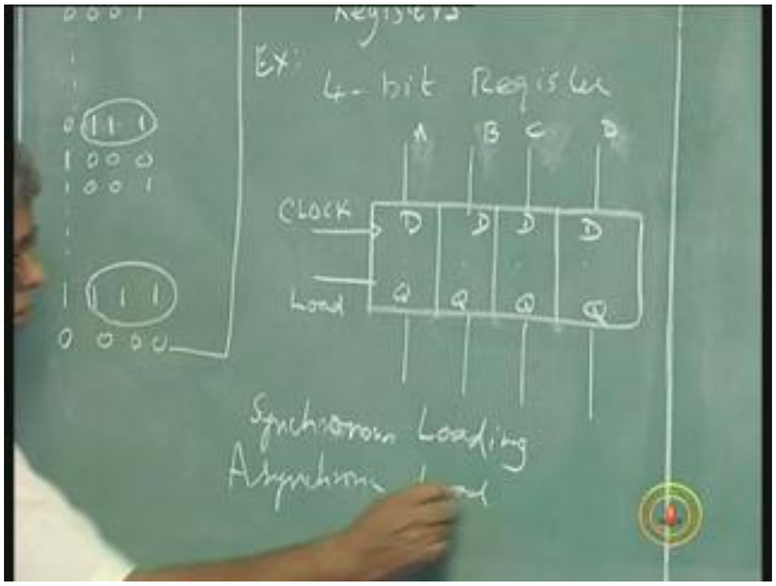
A 4-bit register for example will have four flip-flops let us say they are D flip-flops because I said mostly it is used for data storage and data storage as I said for a D flip-flop is good enough so you have put D D D and D with Qs as the output. Normally these flip-flops can be controlled by single clock. Internally there is a connector of course. As I said externally when we are looking at a device with 4-bits as a user you have only one clock to connect one power supply to connect so all these flip-flops need Vcc and ground or Vdd and ground or whatever it is the nomenclature for the supply voltage. Similarly when we say one clock pulse it can be edged triggered positive or negative or whatever it does not mean that these flip-flops do not require clock it's all internally connected, the clock is connected like this.

Suppose I only see one clock (Refer Slide Time: 24:00) so like that internally they will be connected. likewise here also I will show the clock here but don't think that only the first flip-flop gets a clock it is connected to all flip-flops so I can put any data here or if you want to put A B C we have been using A as the most significant bit and D as the least significant bit so when you put this data A should be the most significant bit. We can just use this this way. Now I can put this information and clock it, it will remain constant it will remain inside and also be available here when I want it I can delete it off, it is a memory. In order to get the inputs into the flip-flops and the register you need to put the data that has to be stored or the inputs, when you clock it these inputs get stored and they are available.

But the problem with this would be I need to keep the data constantly here because the clock keeps coming. You cannot design a clock for every little piece of hardware in your system you are designing a sub system you are designing, a clock is a common entity. Suppose I have a big digital system or a huge digital circuit there are several functional units and many of them require clock pulses for operation. So the clock is generated and distributed commonly to all these units. This clock may be at a frequency which is much faster than the rate at which I want to put the data and remove it.

Suppose I want to put the data and then keep it for a while even when the clock keeps coming and going I don't want to disturb it, so only when I want to use it I will use it and then I will put another data that means I want another control independent of the clock. If I do not have that independent control when I clock it the data gets in and if the data doesn't remain here the data may be available for a short period and when it is available I have to store it, if it is going to be available forever then there is no need for storing it, why should I store something which is always available?

(Refer Slide Time: 29:17)



You need this data but only when it comes you can store it but I don't want the storage to be removed by the next clock pulse because the data has been removed. Data as been removed means it would have taken other values which are non related, the arbitrary values where it can all be 0s, 1s or they can be 0s or 1s or any random combination and next clock pulse will overwrite my data, the next clock pulse will put this random junk thing into my flip-flops so my stored data gets corrupted as I call it, cleared, corrupted, erased or whatever term you want to use.

Therefore I need another control in which I want to store the data when it is available at that time and other than that period the data will not be disturbed so I need a control called load control. Now I am going to make it in the internal circuit such that the clock will be applied when the load is….. That is the data will be stored into the flip-flops only when the load is active and not immediately when the load is active but when the load is active and when the next clock pulse arrives so can do that way. Or I can say simply load will put the data in so I can have two more reserve operations. One is called asynchronous so again we have synchronous and asynchronous.

Synchronous load means the load has to be high and the clock should be activated for it to store the data. Asynchronous means as soon as the load pulse arrives whatever data that is here will be stored irrespective of the clock pulse that is similar to the reset, preset, clear conditions we talked about. We can have a choice of synchronous load or asynchronous load. But the advantage is when the load is removed the store data is here inside. Even when the clock comes at a very fast rate much faster than the rate at which I want to read from this memory, for example I have a clock of one megahertz that is the clock period is one microsecond so every one microsecond we have an edge of the clock whereas I want to put the data here and keep it for hundred microseconds. If load was not there after the first microsecond the data will be there and after the second microsecond it will be removed by and rewritten by whatever is here at that time which may not be the original data, I might have removed, the data must have been there for a short period it just disappeared and A B C D has taken random values and after one microsecond my storage will be junked. So when I look at the data after hundred microseconds when I really need it I will find junk inside and not what I put.

Now with the load I have the control. I can put this load at the time when this data was available and remove this load pulse and then the clock may come and go but then this data cannot enter inside, the new junk value cannot enter in. Even though you don't have the original values they have been removed arbitrary random values that a b c d assumed do not get into the flip-flops until another load pulse comes. So after reading the data after another microsecond I can put one more load pulse and capture one more set of data. Likewise I want to clear a flip-flop that also is possible so I can have one more control called clear. Clear also can be synchronous or asynchronous the same concept, clear removes the data, clear makes the data 0, 0 0 0 0.

Thus I have this independent control and anytime I want to remove the data or make it as 0s I can use a clear input automatically irrespective of what is there on A B C D and all the flip-flops will assume 0s. Again that can be done with the clock pulse or independent of the clock pulse. If it is done independent of the clock pulse it is called asynchronous but when it is done along with the clock pulse where the clear will work only with the clock edge and without that it will not work that’s called synchronous clearing.

As soon as the clear pulse is given if the flip-flops change to 0 0 0 0 independent of the clock pulse if it’s 0 or 1 or an edge then you call it asynchronous clear. So in short I have these registers which are bunches or groups of flip-flops which has several things, I can put the data in and keep it as long as I want and take it of read it of and use it I can again use the same data I can have a data which is stored and read it of again and again that’s also possible because as long as I do not load new data in the data is going to remain and this is not going to go once you read of, there is nothing wrong in using these values that’s all. When you read of a value it does not mean that the data is destroyed, it is not destructive. Or I want to change the value I put a new set of value load it and if I want the data to be 0s I will clear it. These are all very convenient control inputs which will be used in your circuit when you design larger systems. We will see how to use some of these things in our subsequent lectures where we will use some of those designs using these counters.

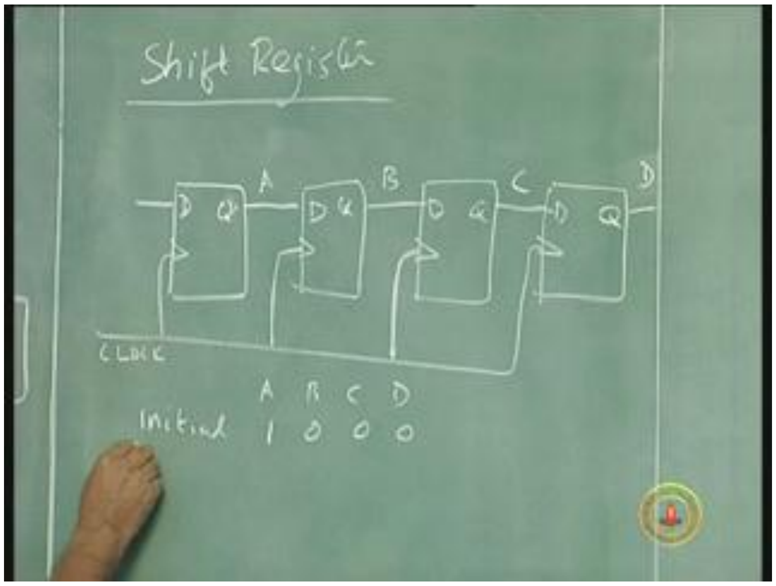
Thus data storage is the major application of flip-flops and counting is one of the applications. Data storage is the major application, division by factor of 2 is another application but data storage is a major application and usually D flip-flops are used for that because you need only one input data and the data can be manipulated by use of clear and the time at which data is put and cleared which is controlled by two inputs called data and clear in addition to the clock which is universal for the whole system and that clock may not be the rate at which we want to put in the data or clear the data.

If it is the same of course you don't need all this but it is generally not true because I may have different adjustments in different places in my system, not all registers get cleared at the same time as frequently as the clock arrives some of them may have to store data for a longer time, some of them may have to store data for a shorter time and none of them may be able to store the data, clear the data at a rate which the clock arrives so the clock frequency is desired by so many other factors.

Therefore we don't want to get into it because the speed of the system will decide the clock frequency, this is only for a short duration may be for a nanosecond or a microsecond the data will be available or useful so that is of no use as good as not being stored. If you have a data which is going to be cleared immediately what is the point in storing it. A variation of this register is called a Shift Register. these are inputs which are applied to all the flip-flops at the same time, clock supplied to all the flip-flops at the same time, load applied to all the flip-flops at the same time, clear applied to all the flip-flops at the same time, and the outputs are available parallely.

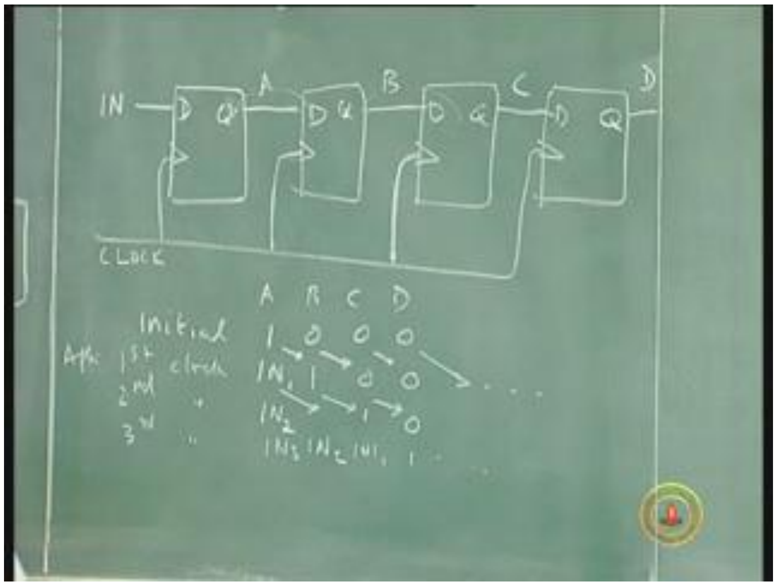
On the other hand if I have a operation where I have a first D flip-flop and the output Q is fed into the second flip-flop and the output Q is fed into the third flip-flop and an output Q is fed into the fourth flip-flop you can have any number of flip-flops, I am just giving you four as an example. And of course I clock them synchronous, hereafter until I specifically say it is asynchronous or repel counter we will use synchronous counter where the clock is applied to all the flip-flops at the same time.

(Refer Slide Time: 38:05)



When we need to use a repel counter specifically I will tell you it’s an asynchronous counter or a repel counter. So this is my clock, and instead of putting the data simultaneously in all these flip-flops now I am connecting the output of the first flip-flop into the input of the second flip-flop, output of the second flip-flop to the input of the third and so forth. Now whatever data I put after every clock pulse that data will shift to the next flip-flop. Suppose initially I put a data I call this initial data A B C D and because this is connected here, this is connected here, this connected here, this one will go here (Refer Slide Time: 38:33) then after first clock pulse, whatever I give as the value I will call this input value IN so this IN will come here. It is a new value and this 0 will be bumped off gone (Refer Slide Time: 39:00).

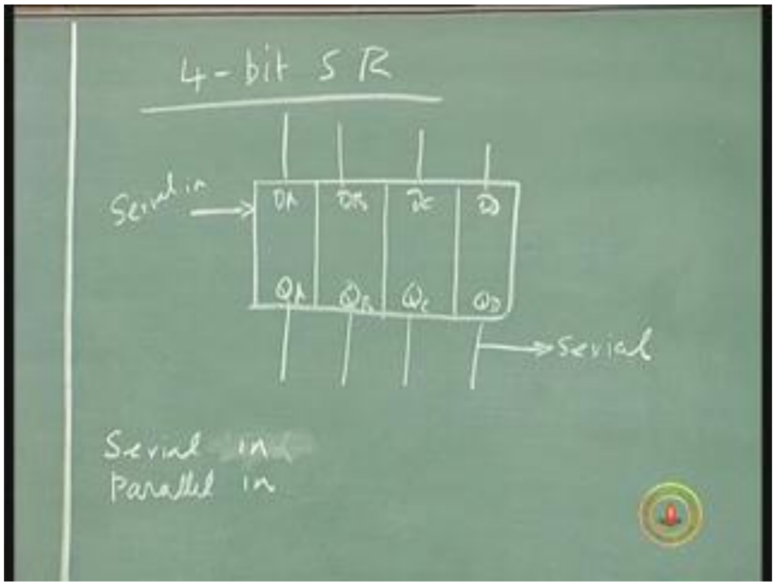
(Refer Slide Time: 39:50)



After the second clock pulse this would have moved here, this would become 1, this would become 0 and what we have here will be IN_2 second input whatever it was and after third it is IN_3, IN_2, IN_1 and 1 and so forth. I keep shifting the data from one flip-flop to the next flip-flop, to the next flip-flop, to the next flip-flop, to the next flip-flop and so on. Such an arrangement of flip-flops is called a Shift Register because it shifts, it's a register in the sense it shows data but it keeps shifting the data from flip-flop to flip-flop to flip-flop to flip-flop it's a Shift Register. So again I can draw this as one block even though we don't have to show all this inside so it is a 4-bit Shift Register.

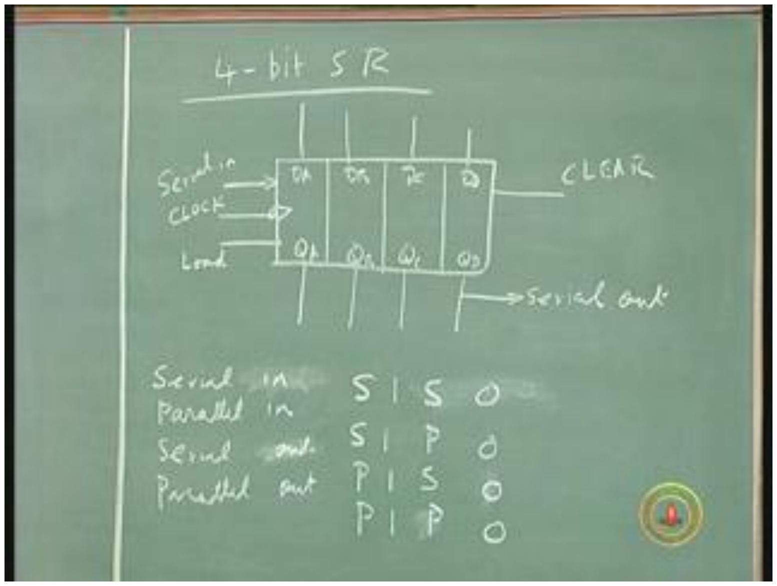
The SR stands for Shift Register in this case. I will not show individual connections and the output of this. I have put the data here, this input is called serial in. The four flip-flops have outputs QA QB QC QD and four inputs called DA DB DC DD. Now QD is also the serial output. I have now various combinations I can load a value of DA DB DC DD all of them at the same time and then start shifting. Shifting is always from one flip-flop to the next flip-flop. But the way in which I load the flip-flops can be one at a time, serially I can put one at a time, initially I can clear all the flip-flops and I don't have to even clear the flip-flops. I start pushing in data one after the other it will go and get loaded whatever way it is. Or at the same time I want to put the all flip-flops at the same time of different values so I can have the option of serial load parallel load or serial input parallel input. Likewise I can redraw the flip-flops QA QB QC QD all at the same time or I can look for the output at serial output last output. So I can have a serial out and parallel out. Then you have a clock which is common to all of the flip-flops as I said in the case of registers.

(Refer Slide Time: 42:30)



So when I put parallel data in how do I get them into the flip-flops? I get them by loading so I need a load which is common to all the flip-flops I can have a clear also which is common to all. So I may have all these features in Shift Register. I can clear all the flip-flops and make them 0 to start with. I can load a new set of values by putting A B C D different values and loading it on by using load. And after clearing or loading if I start the clock then during every clock pulse the clock will shift the output from one position to the next that is the main operation it will do and any point in time you can freeze what you have and read off the values parallely or you want to read serially you have to keep applying clock till you get all the bits out. that means I have four different combinations of operations, I can have serial in, serial out, flip-flops, serial in serial out, serial in parallel out, parallel in serial out, parallel in parallel out these are four combinations are possible.

(Refer Slide Time: 44:27)



I will give input serially take output serially one bit at a time. This operation is a minor operation that is I will keep loading and it gets the output. Whatever I put here comes out here after a delay of the number of clock pulses equal to the number of flip-flops in this case four clock pulses. Whatever data I keep giving here that keeps coming out here after a delay of four clock pulses. So, four is only an arbitrary number I don't have a 4-bit Shift Registers, I can have 8-bit Shift Register or I can have any number. Usually four and eight are available as a product as an IC in the market. So either use a 4-bit Shift Register or 8-bit Shift Register. Hence after four clock pulses or eight clock pulses as the case may be whatever you put comes out so this can be used as a delay element, that's all.

Suppose you want the data to come but then the same data as to go out after a particular delay I can use this serial in serial out mode. Number of the delay is equal to number of flip-flops times the clock period. If there are four flip-flops then it is four clock pulses delay, four clock periods, eight flip-flops eight clock periods delay. Serial in parallel out is when you have the serial input data or I want to take it parallely it is called serial to parallel conversion. What happens is in some applications I have a serial input data coming in but I have to wait for the whole bit.

For example, let us assume ASCII code 8-bits, I have an 8-bit Shift Register I get an ASCII code from some other device unfortunately that device can only transmit one bit at a time, unfortunately that device which is supplying the ASCII that keyboard is only a serial connection so the 8-bits of the ASCII including the parity bit keep coming like one bit for every clock period but I cannot process this until I know what ASCII code it is, I cannot process until the ASCII code is completely built then I can store it or do whatever with it. This is an application where serial input parallel output parallel output is required for processing whereas serial input is the limitation of the device like a telephone modem. You know that in a telephone you have the dial up modem, the data comes serially because you have a pair of lines, the telephone has a pair of lines you use it in your computer, it is the limitation in your telephone line not the generation of data, I want to send a mail to you, my computer can also give you a parallel output your computer can also process in parallel but in between the medium is serial, telephone line. Therefore I have to convert my parallel data into serial mode data and send it to you that is what my modem does and your modem gets it back into parallel. So these two things are covered; serial input parallel output and parallel input serial output, examples are modems.

Finally this parallel in parallel out is similar to the register wherein I want to put in the data and take it out that's all, I load it I take it. If you disable the load it will remain as long as you require so this is same as the register operation, shift has no meaning in this. If you want to put the data and take it out as it is where is the question of shifting it there is no need for shifting. My Shift Register becomes a normal register operation in this mode. It's a very very powerful device as I said it can be used for many things such as for storing, for delaying, for converting parallel data into serial, converting serial into parallel and I only talked about shifting to the right.

By a proper connection the output of this flip-flop can be connected to the input of this flip-flop, the output of this can be connected to input of this and the output of this can be connected to this, is it not possible? The output of this I connect to this output of this connected to this and the output of this is connected to this (refer Slide Time: 49:58).

Suppose you are taking output of this and connecting to this, output of this and connect it to this output of this and connect to this then this becomes a left shift flip-flop. So I can have a Shift Left Shift Register and Shift Right Shift Register. So a Shift Register can have a serial input as data input, parallel data input, serial data output, parallel data output it can shift left, shift right.

You can buy an appropriate Shift Register you can buy a shift left 4-bit Shift Register with serial input and serial output or you can have a register which does not need a shifting. You have got all these features you don't which one you will use, I want shift left feature, shift right feature, clearing feature, loading feature, serial in feature, serial out feature, parallel in feature and parallel out feature so all of them I want but I don't know which one I will use either a 8-bit Shift Registers or 4-bit Shift Registers such Shift Registers are called Universal Shift Registers because you can do all things that you want to but of course not all of them at the same time but you have to configure it for one particular operation either a shift left operation or a shift right operation or parallel load and serial out, serial in parallel out or whatever. These are available in market commercially but you should know what is inside. Inside it is after all nothing so I have reduced the width of all these registers, counters and all that you have. There are two chapters in your book of flip-flops, registers, counters so it is a basic storage element of a latch connected to a clock and you can use flip-flops in so many different ways to get all those different things similar to what we did with gates, in gates we had so many other applications we called them adders, you called them parity generators, parity checkers and all that.