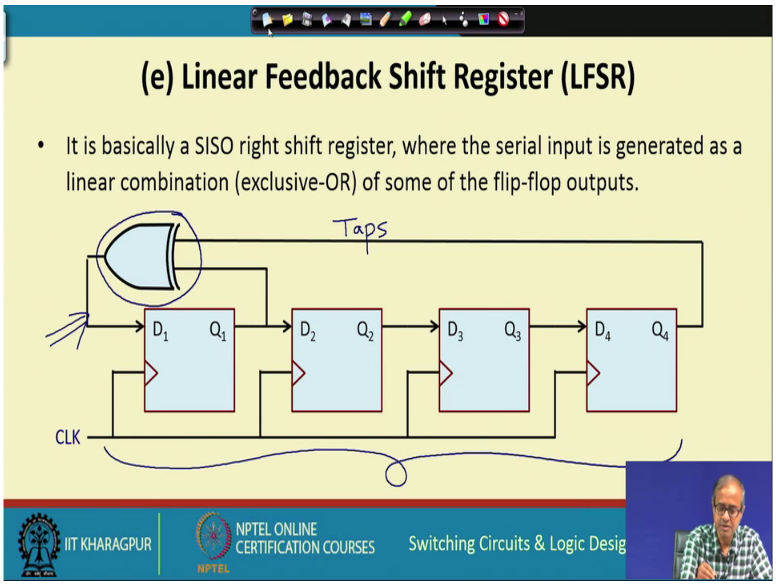
# Switching Circuits and Logic Design

**Prof. Indranil Sengupta**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kharagpur**

## Lecture – 44
## Design of Registers (Part – III)

So, we continue in this lecture with the discussion on registers which we continued in the last couple of lectures. So, if you recall we discussed some of the different kinds of registers namely the parallel in parallel out or PIPO registers and some variety of shift registers. So, we continue with our discussion in the lecture, Design of Register, the third part.

(Refer Slide Time: 00:47)



### (e) Linear Feedback Shift Register (LFSR)

- It is basically a SISO right shift register, where the serial input is generated as a linear combination (exclusive-OR) of some of the flip-flop outputs.

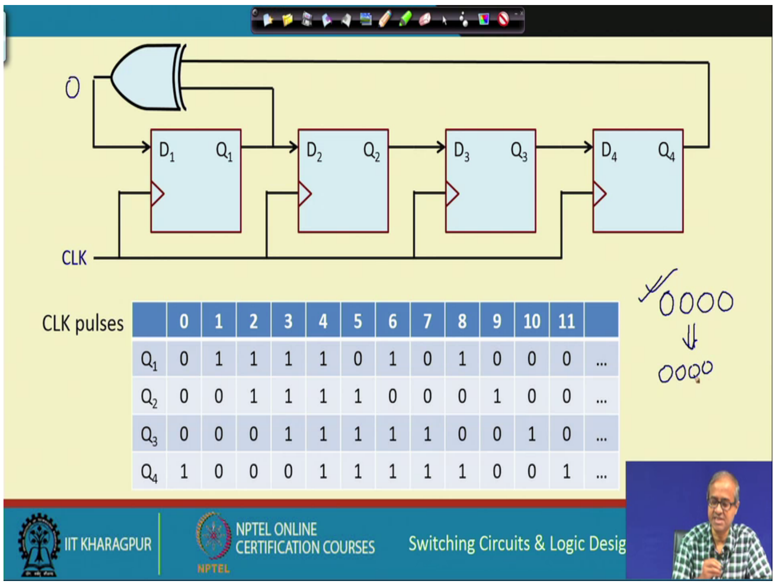
Taps
D1
Q1
D2
Q2
D3
Q3
D4
Q4
CLK
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desig

So, we start with I means another variation of the shift register we discussed four different types of shift register in the last lecture namely, ring counter, Johnson counter, bidirectional shift register and universal shift register.

Now, here we consider another kind of shift register which is very useful in many applications namely linear feedback shift register. Linear feedback shift register pictorially it looks like this. So, if you look into this there is one part of it which looks like a conventional shift register, you forget the XOR gate for the time being the four D flip-flops connected in chain in cascade, this looks like a normal right-shift shift register, right. And, we call it a linear feedback shift register.

Now, I am not going into the formal definition of linearity, but it would suffice if you remember that this exclusive-OR function is linear. So, in a Linear Feedback Shift Register or LFSR in short which you call the idea is that we have an exclusive-OR gate to the input of this exclusive-OR gate we feed some of the outputs of this LFSR. Here for example, we have taken this Q 1 and Q 4, these are called the taps. We take some taps from the output of this chip register and feed it to the input of this XOR gate and the output of the XOR gate is feeding here the D 1 input of the first flip-flop. So, the input is not coming from outside rather this exclusive-OR gate is generating the next input that is to be shifted. Basically, a linear feedback shift register looks like this.

(Refer Slide Time: 03:01)

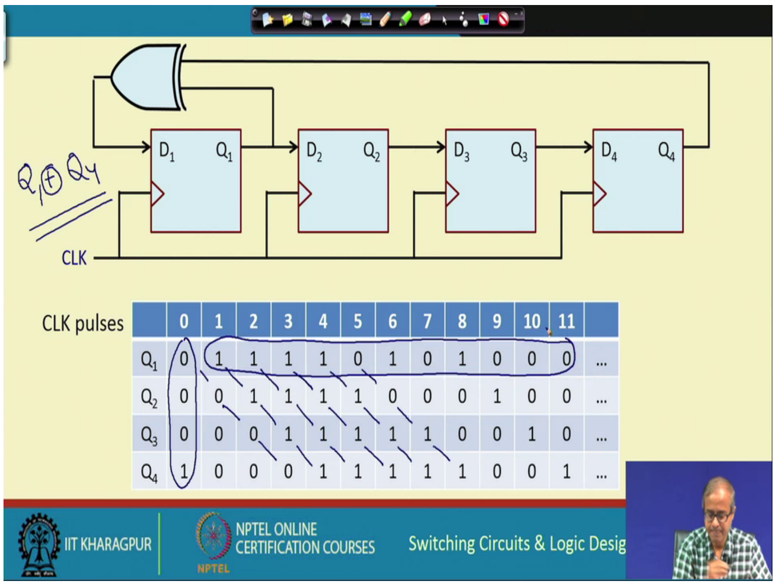


| CLK pulses | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_1$ | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ... |
| $Q_2$ | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | ... |
| $Q_3$ | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | ... |
| $Q_4$ | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | ... |

Let us look at the kind of patterns that are generated by this LFSR. The same example LFSR I am showing here, but one thing you observe that if the initial state of the LFSR is all 0, let us say 0000 then the inputs of this XOR gate will also be all zeros and if it is all zeros the output of the LFSR will also be 0 and 0 will get shifted in.

So, this 0000 after one clock will remain 0000. So, it will never change. So, in a LFSR we should not initialize the register to the all zero state because if you do so, the state will never change. Now, what we do? We initialize it with any other or some other suitable nonzero state in the example that I have shown here. Here I have shown the clock pulses, 0 means this is the initial state. So, I am assuming initially Q 4 is 1 and the other three are zeros.

(Refer Slide Time: 04:22)



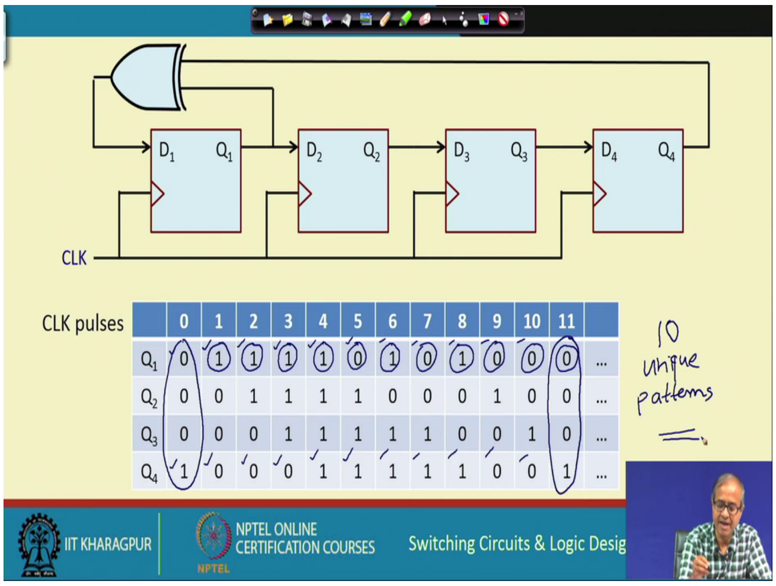
| CLK pulses | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_1$ | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ... |
| $Q_2$ | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | ... |
| $Q_3$ | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | ... |
| $Q_4$ | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | ... |

Now, if we look at the way we have connected basically D 1 is nothing, but Q 1 exclusive-OR Q 4. So, you take the exclusive-OR of Q 1 and Q 4 and whatever it is, that is the next bed that will be shifted in. So, in this LFSR the first thing is that it is a shift register. So, this shifting is taking place just like a normal shift register. So, with the clocks the bits are getting shifted like this.

Now, the point to notice that what is the next bit that is shifted in out here. Let us look into this right. The first bit this one this will be the exclusive-OR of 0 and 1, Q 1 and Q 4, 0 1 is 1. So, 1 get shifted in, the other three bits are simply shifted. Next one this is also the exclusive-OR of this 1 and 0, then again exclusive-OR of 1 and 0 is 1 exclusive-OR of 1 and 0 is again 1, XOR of 1 and 1 is 0, XOR of 0 and 1 is 1, 1 and 1 is 0, 0 and 1 is 1, 1 and 1 is 0, like this it goes on 0 and 0 is 0, 0 and 0 is 0 and so on.

Now, see that we started with this state 0 0 0 1 and after 10 clock pulses at the 11th clock pulse we again get back 0 0 0 1.

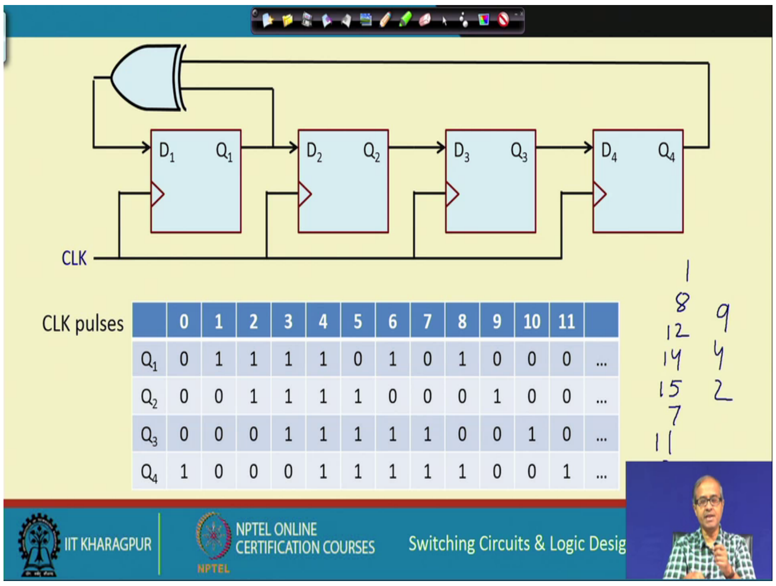
(Refer Slide Time: 06:13)


D₁
Q₁
D₂
Q₂
D₃
Q₃
D₄
Q₄
CLK
10 unique patterns
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desig

| CLK pulses | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_1$ | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ... |
| $Q_2$ | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | ... |
| $Q_3$ | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | ... |
| $Q_4$ | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | ... |

So, in this example there are 10 unique patterns that are generated, 10 unique patterns. Now, one property of this LFSR is that the patterns which are generated they bear good randomness properties.

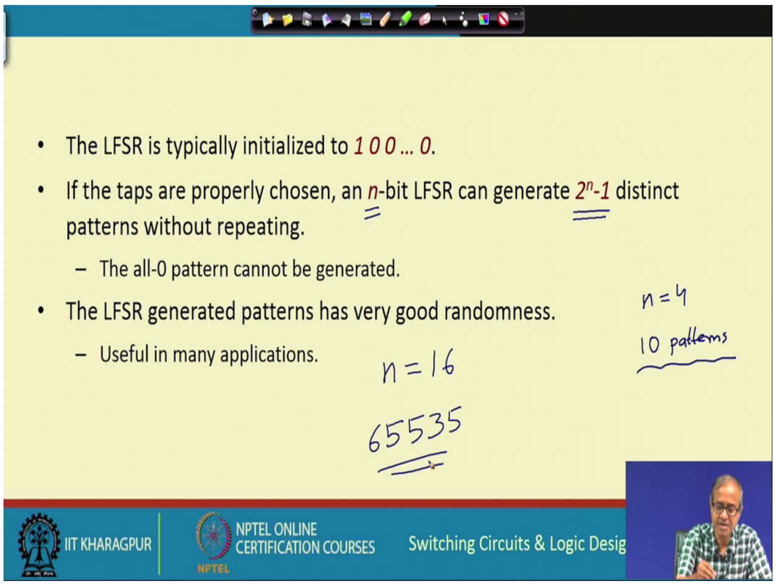
(Refer Slide Time: 06:46)


D₁
Q₁
D₂
Q₂
D₃
Q₃
D₄
Q₄
CLK
1
8
12
14
15
7
11
9
4
2
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desig

| CLK pulses | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $Q_1$ | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | ... |
| $Q_2$ | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | ... |
| $Q_3$ | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | ... |
| $Q_4$ | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | ... |

Like for instance if you treat this 4-bit output of this LFSR as the as the binary number and if you look at the decimal equivalents for example, in 0 0 0 1 means 1. The 1 0 0 0 means 8, 1 1 0 0 means 12, 1 1 1 0 is 14, 1 1 1 1 is 15, 0 1 1 1 is 7, 1 0 1 1 is a 11, this is 3. Then, 1 0 0 1 is 9, then 4, then 2, you see this numbers are very random. There are some standard tests of randomness and it can be shown that this LFSR generated patterns they exhibit good randomness properties with the exception of one test because the bits are shifted the correlation between the bits that is generated in one stage and the next stage will be just shifted 1-bit in time other than that all other randomness properties are very well satisfied.

(Refer Slide Time: 07:47)



- The LFSR is typically initialized to *1 0 0 … 0*.
- If the taps are properly chosen, an $\underline{n}$-bit LFSR can generate $\underline{\underline{2^n-1}}$ distinct patterns without repeating.
  - The all-0 pattern cannot be generated.
- The LFSR generated patterns has very good randomness.
  - Useful in many applications.

n = 4
10 patterns
n = 16
65535
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desig

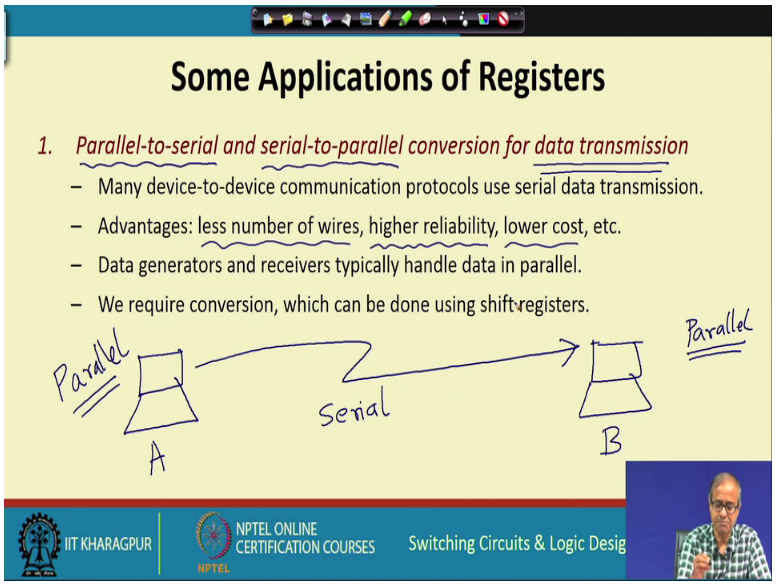
So, as we have seen at in a practical application LFSR is typically initialized with one 0 and the rest 1's and the point to notice that in the example that we took it was a 4-bit LFSR and it generated 10 patterns, right. It generated 10 patterns, but if we choose the tapping points in a suitable way then and n-bit LFSR can generate 2 to the power n minus 1 distinct patterns, well if I minus 1 with the exception of the all-0 pattern that we have said because all-0 pattern cannot be a state in a cycle because if we initialize the LFSR to all-0 pattern it will indefinitely remain in the all-0 state, it will never come out.

So, by suitably choosing the taps we can generate a very large number of distinct patterns. For example, if I have a 16-bit register LFSR then 2 to the power 16 minus 1 is 65535. So many distinct random patterns can be generated, right.

Now, there are many applications where such random patterns are used. Well, we shall be discussing one such application later in this class in this course, namely while we are testing a digital circuit for faults fault testing. There also this randomness properties are very well I means exploited in some of the techniques this we shall see later.

Now, let us look at some of the typical applications of registers you have seen the different kinds of registers. Registers are used almost everywhere in digital system design. In any digital system design which are sufficiently complex you will find one or more registers inside the system. Let us take a couple of examples, a broad examples.

(Refer Slide Time: 10:06)



## Some Applications of Registers

1. Parallel-to-serial and serial-to-parallel conversion for data transmission
   - Many device-to-device communication protocols use serial data transmission.
   - Advantages: less number of wires, higher reliability, lower cost, etc.
   - Data generators and receivers typically handle data in parallel.
   - We require conversion, which can be done using shift registers.

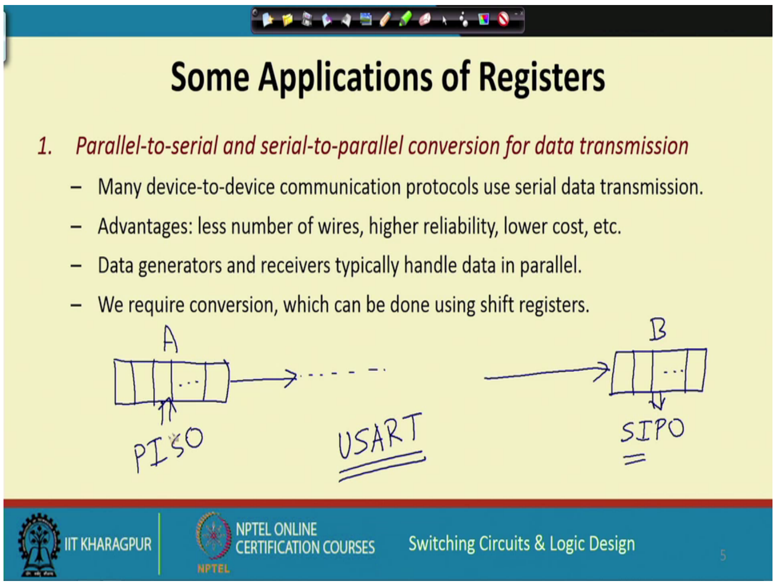
Parallel
A
Serial
B
Parallel
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desig

The first example that we take is that of parallel-to-serial and serial-to-parallel conversion, this is used for data transmission. Let us try to motivate why we need this. Imagine that you have a computer here computer system A and there is another computer system out here, this is B. We want to transmit some data from A to B. Well, B can be A computer it can be some other peripheral device also like a printer etcetera, but you are required to transfer some data.

Now, inside A or B data representation is parallel data are stored in parallel in registers, but when you transfer the data because of various cost considerations you normally transmit data in a serial fashion. Serial fashion means there is a single wire over which the bits are transmitted 1 bit at a time serially. The advantage is that the cost of the cable gets reduced because instead of transmitting 16 bits at a time we will be needing 16 wires for that, but here you will be needing only 1 wire to carry the data. So, the cost of the cable become simple, the protocol become simple, the interfacing hardware also become simple.

So, although inside A and B you have parallel representation of data what was saying that while you are communicating your communicating in a serial fashion. So, there is a necessity to convert this parallel data into serial form and again and the receiving end serial to parallel form. Now, this motivation I have already mentioned many such communication required serial communication, advantages are less number of wires and because the less number of wires the chances of loose connections and other faults are less, it will lead to higher reliability and of course, lower cost, ok. And, this data conversion parallel to serial and serial to parallel can be very conveniently done using shift registers. How? Let us look at it.

(Refer Slide Time: 12:43)



## Some Applications of Registers

1. *Parallel-to-serial and serial-to-parallel conversion for data transmission*
   - Many device-to-device communication protocols use serial data transmission.
   - Advantages: less number of wires, higher reliability, lower cost, etc.
   - Data generators and receivers typically handle data in parallel.
   - We require conversion, which can be done using shift registers.

A
B
PISO
USART
SIPO
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Design

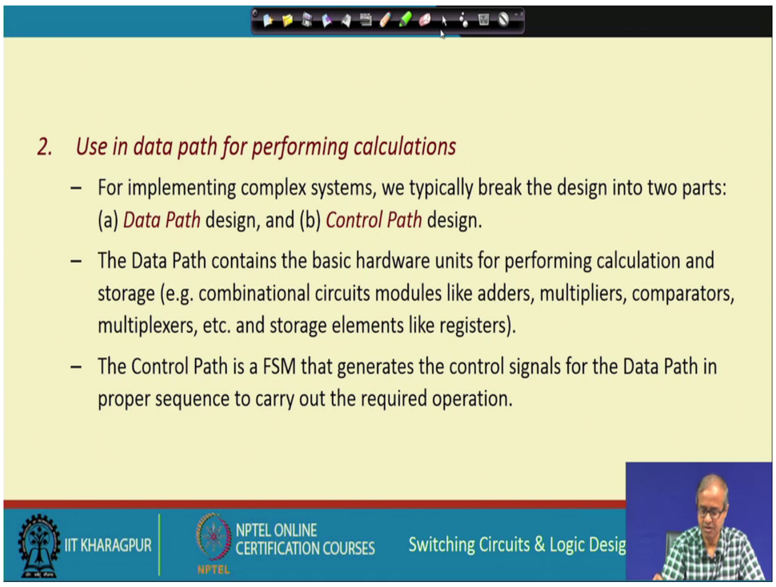
Let us say at the end of the transmitter, this is A the data is stored in parallel in a register and similarly at the receiving end B the data will be received in a register because inside A and B all data processing are done in parallel. But now what you saying is that let this register be a parallel in serial out register and at the other side this B will be a serial in parallel out register. So, what does this mean? So, here this data you can load into this register in parallel. So, this will be something like a universal register universal shift register, you can load it in parallel then you will be shifting them out serially bit by bit, parallel in serial out. So, the bits will be transmitted serially.

And, the other side the data will be entering this register in a serial fashion serial input and after all the bits have been entered you can read out the data in parallel, parallel out.

So, this kind of interfacing hardware is present in almost any communication system in the inside the computers that we see today and these devices are normally called universal synchronous asynchronous receiver transmitter. If you look at the literature you will see that there is something called USART. USART, Universal Synchronous Asynchronous Receiver Transmitter. Now, inside this u USART this kind of PISO and SIPO registers are there; for transmission you need the PISO mode, for receiving unit this SIPO mode.

This is how data transmission and receiving takes place during data communication serially. So, this is one of the very important applications of registers specifically shift registers.

(Refer Slide Time: 15:10)


2. *Use in data path for performing calculations*
   - For implementing complex systems, we typically break the design into two parts: (a) *Data Path* design, and (b) *Control Path* design.
   - The Data Path contains the basic hardware units for performing calculation and storage (e.g. combinational circuits modules like adders, multipliers, comparators, multiplexers, etc. and storage elements like registers).
   - The Control Path is a FSM that generates the control signals for the Data Path in proper sequence to carry out the required operation.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
Switching Circuits & Logic Desig

Let us look at another very important application of registers, but here we are not talking about shift registers, but normal parallel in parallel out registers. There are many designs where we need registers to store some temporary data and there you need a required this kind of PIPO registers. Let us take a couple of examples to illustrate how such designs can be carried out and how this registers can be used there, right. So, this application talks about something called data path.

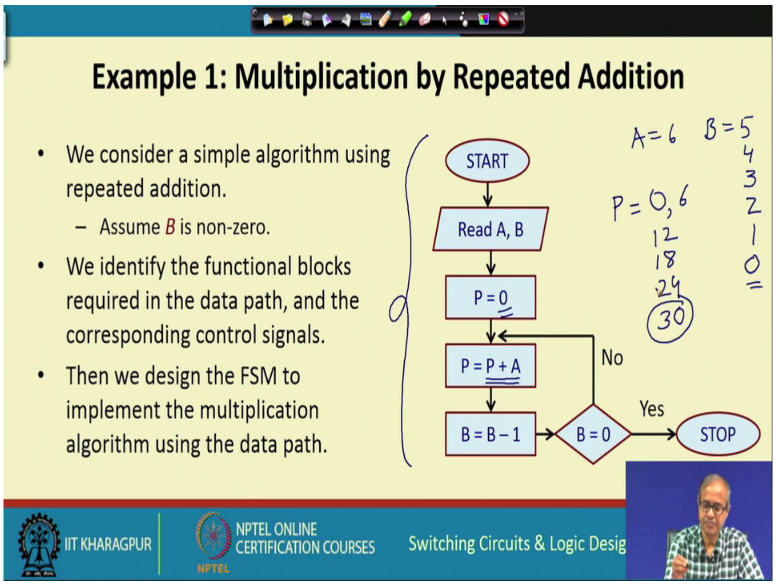
Registers can be used in data path for performing calculation we shall explain what is meant by data path. The basic idea is while you are designing a complex system, relatively complex system; well, normally for small systems what to do as we have seen if it is a sequential circuit you break in up into some kind of specification in the terms of a state table or a state transition diagram, then following some systematic steps you arrive at the final design. But, let us take an example suppose I want to implement circuit that compute the greatest common divisor GCD of two numbers. You see designing such a circuit using this FSM mechanism will be extremely difficult. There should be some other simpler method to approach this kind of complex problems. So, any kind of higher level description like this requires a different approach.

The approach says that you need to separate out the data path and control path of the design. So, what are data path and control paths? Data path will contain all the basic hardware units that are required for the calculation. For example, the example that I mentioned GCD calculation, for GCD calculation you will be requiring some registers to store the input numbers, some register to store some intermediate data, some adders subtractors, maybe some comparators, these are the kind of hardware will be requiring to carry out step by step calculation for computing GCD, this will constitute the data path of the design, right.

So, it will contain the basic hardware units and they are required for carrying out calculation and also storage. So, some examples are mentioned here some combinational circuit modules like adders, multipliers, subtractors, comparators, multiplexers, and storage elements like flip-flops and registers, right and you need separate unit called control path. The idea is like this the data path contains the basic circuits that are required to carry out the calculation, but someone has to tell what calculation has to be done in what sequence of time. So, there must be some other circuit which will be generating the signals in proper sequence to carry out the required operations, that is the role of the control path.

So, control path is nothing, but a finite state machine. Control path is a finite state machine that will be generating control signals which will be activating the data path. Like for example, if there is a register you can say that you clear the register, load the register, like that. So, this we shall be explaining with the help of some examples. So, that you have an idea that how these are actually done.

(Refer Slide Time: 19:26)



## Example 1: Multiplication by Repeated Addition

- We consider a simple algorithm using repeated addition.
  - Assume *B* is non-zero.
- We identify the functional blocks required in the data path, and the corresponding control signals.
- Then we design the FSM to implement the multiplication algorithm using the data path.

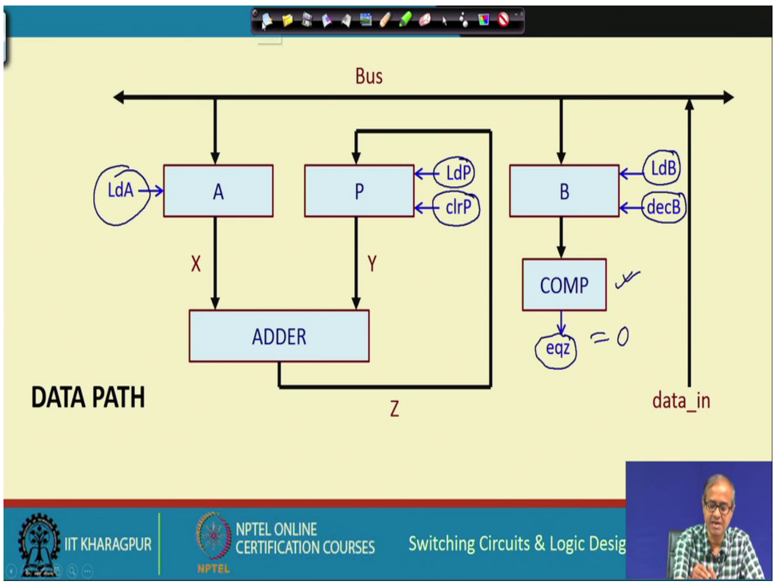
So, let us take one example which is quite simple in terms of the complexity. Here we are saying that let us try to multiply two numbers by repeated addition. So, what do mean by this? Suppose, I want to multiply 6 by 5, 6 into 5 so, the simple approaches I can add 6 to itself 5 times 6 plus 6 plus 6 plus 6 plus 6 that is one way to carry out multiplication. Of course, it may not be very efficient. So, if the numbers are very large we will have to add so many times, but this is a very simple method.

Let us try to see how this can be designed, right. So, here I am assuming that the second number is nonzero, because here we are not checking whether the second number is 0 or not this steps are explain in flowchart form like this the two numbers are A and B, these are first given and P is the final product which is initialize to 0. And in a loop we continuously add A to P, P plus A and the result we are putting it back to P and after every addition we are decrementing B and your testing whether B it has reached 0 or not, if it is not 0 we go back, right.

So, as it said if A is 6 and B is 5 then first P is initialize to 0, first we make P equal to P plus A, A is added. So, P become 6, B is decremented by 1 B becomes 4 B is not 0, no, you again go back again P equal to P plus A, you add A to P. So, now, this becomes 12 decrement B again it becomes 3, B 0 no, go back, again add a 18 decrement B, not 0, go back. So, again add 24 decrement B not 0. So, you so again add 30, decrement B, now B become 0. So, B become 0, you come out and whatever is in P that is your final result, right.

Now, it is up to this designer to decide what are the things you require to implement this flowchart. You see intuitively speaking what are the things you need? You need two registers to store A and B, you need one register to store the product P, you need an adder circuit. Well, B equal to B minus 1 you can you can use a subtractor, but because you are just subtracting by 1 we shall see later that we can use a counter to do it. A counter that can count upper countdown, you can use a down counter also decrement by 1. So, the register B that we use that can be a down counter and a comparator that will check B with 0 that will constitute the data path and the control path will be generating signal such that competition is carried out in this sequence as specified by the flowchart.

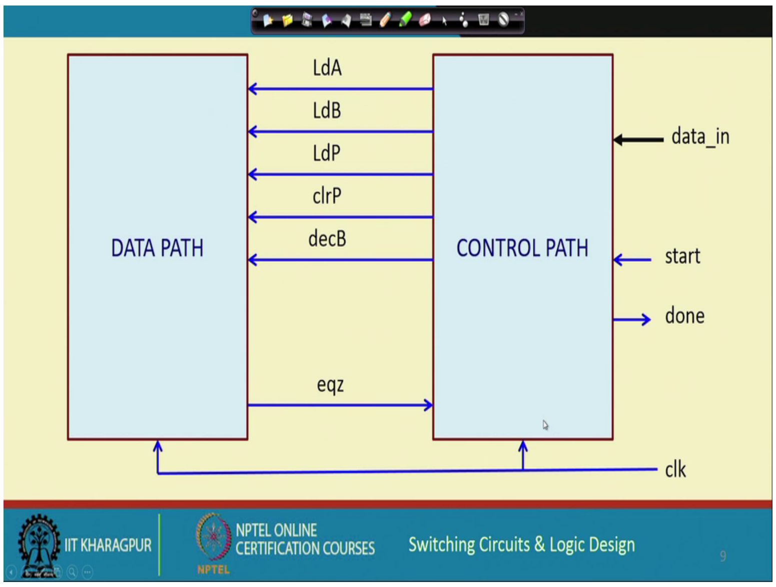
(Refer Slide Time: 23:10)



Let us see, this will be the data path how it will look like. Just see you have a register A here, register B here, just look at it in a little more detail. This register A has one control input can load A, which means this is a parallel in parallel out register. So, from outside data in you can load a data into A the new data. Similarly, you can load A data into B for that also a load signal is there load B and you may also required to decrement B by 1 for that there is another control signal decrement B and P is the partial product which initially we will have to set it to 0. So, there is a clear control clear P it will said it to 0 and every time A plus P is computed and the result goes back to P for that you will have to load P again and there is a comparator circuit which will be generating and output which will tell whether B is equal to 0 or not, ok.

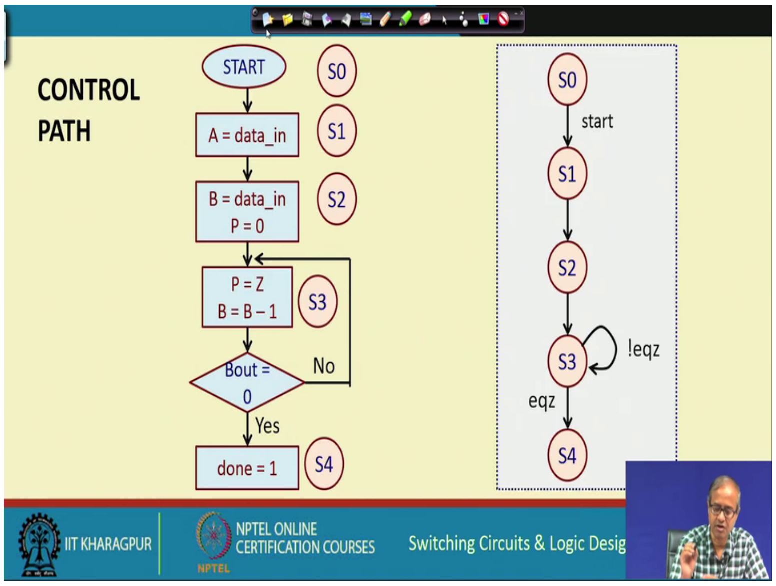
This is basically the data path which you require to carry out the JCD computer the computation of multiplication, but in what sequence there the control path will come into the picture.

(Refer Slide Time: 24:45)



So, for the control path the idea will be like this, there will a data path like this and there will be a control path like this. The signals I have already mentioned. These are the control signals with the data path with the control path will be generating and this equal to 0 signal will be an input for the control path and these are some external signals start, done and a clock is coming.

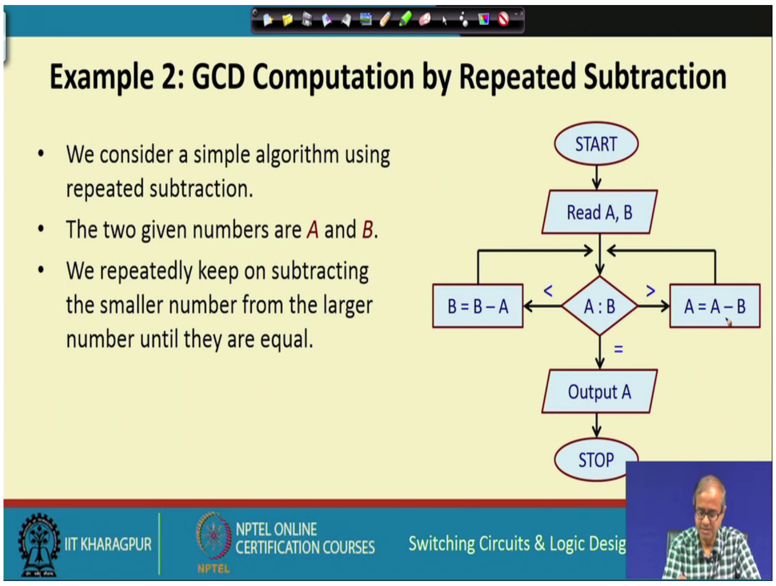
(Refer Slide Time: 25:07)



The control path I am not showing the complete design I am showing you this will be an FSM. This is the same flow chart I am drawing in a slightly different way, these are the different steps of computation that you need to carry out and S0 to S4 are the 5 states. And, this you can encode as a finite state machine like this where sequentially will go for S0, S1, S2, S3 and S3 will be remaining in this loop until this equal to 0 condition is holding true, then you come out.

So, here I have shown you simplified state transition diagram the idea is that whenever you are in a particular state for example, in S2 now whenever you are in S2 you have to generate the control signals for loading data in B load B and clear P, P equal to 0. Similarly, when you are in state S3 will have to load the value of P; that means, load P and decrement B and so on, this is how it will work.

(Refer Slide Time: 26:19)



## Example 2: GCD Computation by Repeated Subtraction

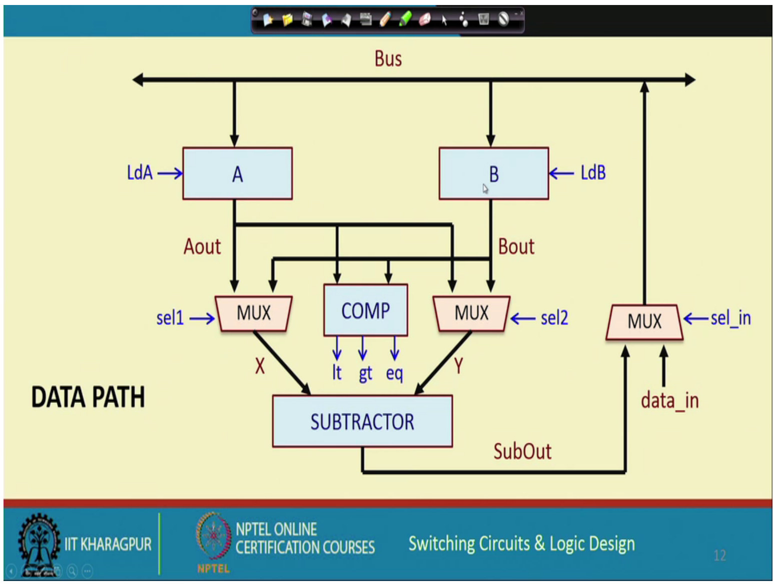
- We consider a simple algorithm using repeated subtraction.
- The two given numbers are *A* and *B*.
- We repeatedly keep on subtracting the smaller number from the larger number until they are equal.

START
Read A, B
A : B
<
>
=
B = B – A
A = A – B
Output A
STOP
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Switching Circuits & Logic Desig

Let us take another example this example is somewhat similar, but a little more complex we want to compute the GCD of two numbers greatest common deviser by repeated subtraction. So, what is the basic approach? The two given numbers and A and B. So, we repeatedly keep on subtracting the smaller number from the larger number; that means, you have two numbers A and B, you compare the two numbers A and B, if A is less than B then you do B equal to B minus A. If A is greater than B you do A equal to A minus B, but if there equal you are done you can output either a or B as the final result.

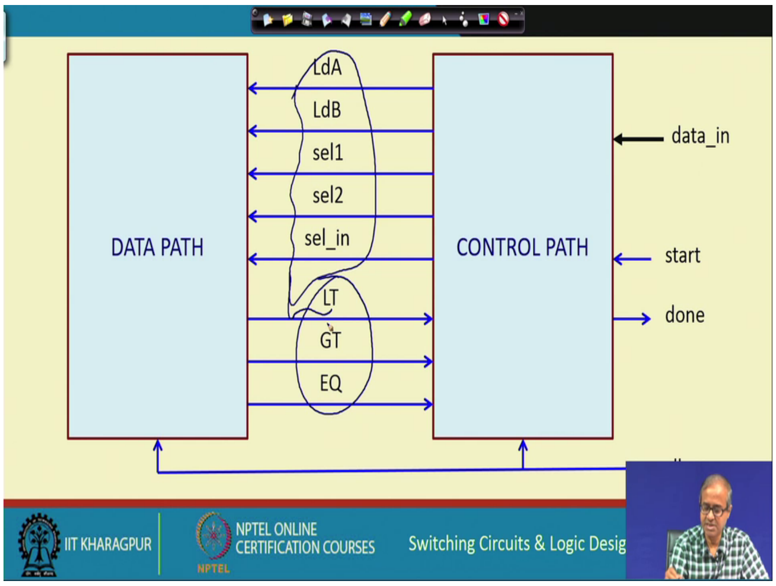
So, you see for this you need again two registers A and B, you need a comparator and you need a subtractor and of course, you need some other circuit because sometimes you are doing B minus A, sometimes you are doing A minus B.

(Refer Slide Time: 27:27)



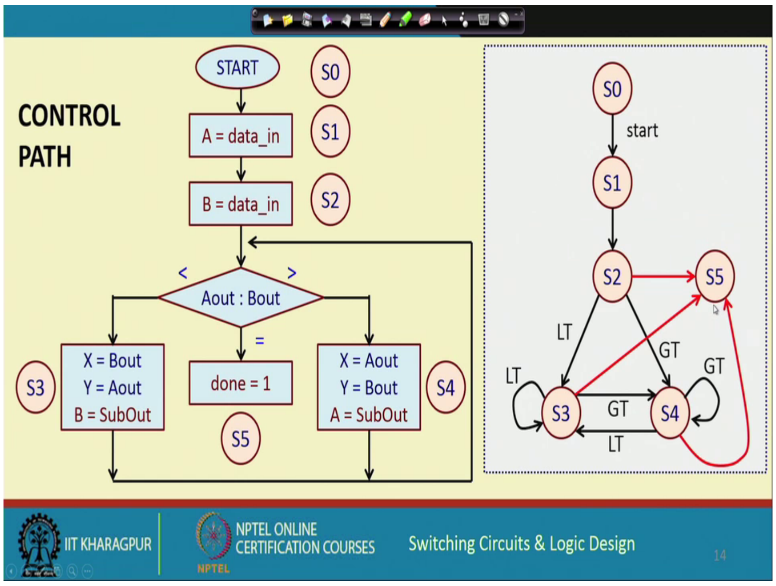
So, possible data path can be like this; the two registers A and B this multiplexer will select whether you are doing A minus B or B minus A. The first input of the subtractor can be either A or it can be B, this multiplexer will be selecting that. Similarly, the second multiplexer will be selecting the second input of the subtractor sorry it can be either A or again it can be B. This comparator will be comparing the values of A and B whether they are less than greater than or equal to and this multiplexer will be either loading the data in A and B from outside data in or during operation the output of the subtractor will be again loaded back into A or B depending on whether it was less than or greater than. So, this is the data path for you can see you are requiring three, two registers in this case and some other combination blocks.

(Refer Slide Time: 28:30)



In a similar way the block level diagram will look like this, data path is here where these are all the control signals which are required and these are the signals which are generated by the comparator less than greater than equal to based on that the control path is taking the decision.

(Refer Slide Time: 28:56)



So, the flowchart for the control path is again shown here. Same flow chart in a slightly different way in terms of the operation with respect to the data path I have shown. Here you can see there are 6 states. So, I recommend you just compare this with respect to data path and verify so, whether this operations are being carried out in a correct sequence or not. And, with respect to the FSM, now the FSM will look like this S0, S1, S2 then depending on this comparison it can either go to S3 or S4 or S5; S5 means you are done and from S3 also after the subtraction when you go back in next comparison you could go to S5. Similarly, from S4 you can also go to S5. So, this is red lines show you the different paths where I means if it indicates that your computation is over, ok.

So, the idea is very simple I mean you start with the flow chart you decide what hardware you need to implement that flow chart the basic operations that are mentioned there, that is your data path that the same flowchart you translate into some kind of a finite state machine specification. Here I have not shown how to design that FSM it is fairly simple because the FSM structure is fairly simple, you already know how to do it, from there you design the FSM and the FSM will be generating control signals for your data path which will allow you to execute the final operation.

So, with this we come to the end of this lecture. Means over the last three lecture we discuss the various kind of registers and in this lecture we looked at a couple of applications. So, we shall be continuing with a discussion in the next lecture where you shall be starting on discussions on the design of counters.

Thank you.