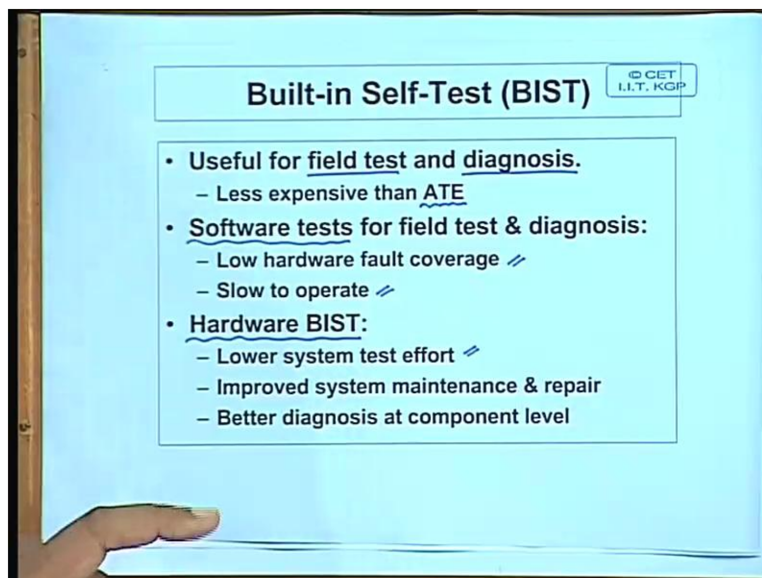
Electronic Design Automation
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture No #34
Testing - V

Last class if you recall we are talking about the various design for testability techniques which make life easier for the test engineers in terms of the effort they have to put in for testing. Now in the structure DFT techniques that we have talked about we talked about techniques like scan chain and partial chain which are very popular methods which are followed in the industry. Whereby given a design we add scan chain to it with the net result is that the resultant circuit whatever we get is much easier to test okay. There is another general philosophy which is also wide spread in use. That is called built in self-test. Well here we are trying to add some more circuit tree inside the chip so that the chip can test itself. But there are number of issues that it to address here. So in the lecture today we will try to address some of these issues which are posed by the best built in self test techniques that are used. So first let us try to see what is this built in self-test technique.

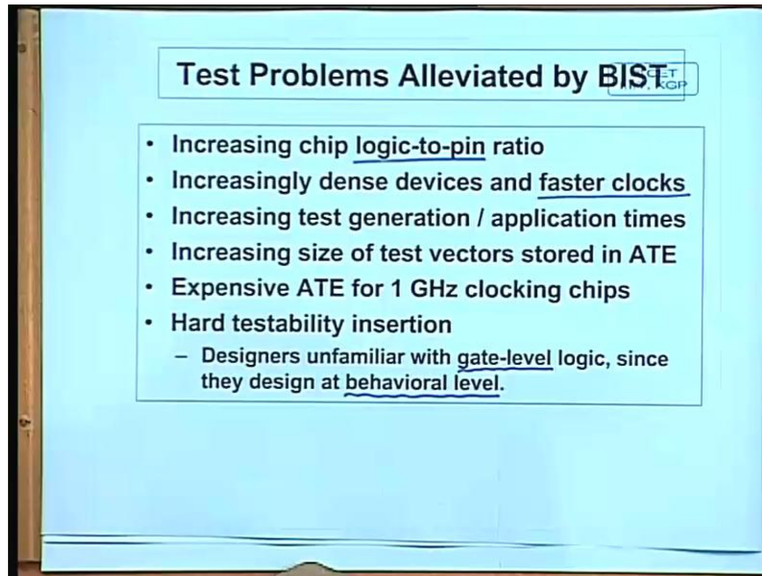
(Refer Slide Time: 02:24)



And what does this involve? Well built in self-test is useful for field test and diagnosis. Field test means in the exact environment where you are using the chip in terms of the application you can test the chip there itself. In contrast if you are using design for testability techniques then the chips have to be tested in a special laboratory where you have an automates test equipment installed. And since this ATE is very expensive equipment, so having based and using it for testing is in general much less expensive. Now talking about field test and diagnosis, so earlier many people used some kind of software tests or diagnostics to test the modules are this sub system so which are working correctly or before just filled. But the problem with software test is that the hardware fault coverage is not that high.

And moreover since these are software programs which you are running they are relatively slow to operate. So a better approach is to have some special hardware mechanism using some special purpose hardware you can have this built in self-test mechanism. So if you have this the system test effort will be much lower because already they are some hardware inside the chip that will take care of almost everything. Because of this the system maintenance and repair activities will be much better. And you can have better diagnosis at the component level. Because if a component fails the component itself will announce that well I am bad okay. So now let us see that what are the major test problems that tackled or handled in base techniques which otherwise would pose the big problem to the test engineers okay.

(Refer Slide Time: 04:33)



So the major test problem that I elevated by BIST are as follows. Well you know that the numbers of components in a chip are increasing day by day. But the number of pins of the chip is not increasing in the same proposal. So increasing chip logic to pin ratio is a major concern. This means that the controllability and observability of the logic inside the chip is going down every day. So if you have some special purpose hardware inside the chip to test the logic inside. So it will be much beneficial. Well this is related point increasingly dense devices and of course faster clocks. This is one important issue. Nowadays we run these circuits are much higher clock frequencies. But in techniques like design for testability DFT scan path.

There we really do not test the circuit at its full speed or full frequency because we need to scan in the data in the scan change and then only we can apply a test vector and we can see the output. So the frequency at which we are actually operating the circuit is much lower than the actual intended operating frequency. But in BIST we can do the testing at the actual fast clock rate and the increasing test generation and test application times that can also be elevated by BIST because it is done automatically inside. And since we are using automated test equipments and the size of the test vectors required is increasing day by day. So the memory requirement of ATE

is also increasing. So these problems are also addressed. And ATE is which work at one gigahertz clocks are higher are very expensive.

This is what you show. And testability insertion with respect to DFT is not always very easy because designers normally designer work at the behavioral level. So the CAT tools automatically we will translate the behavioral level into gate level. So when we talk about inserting some testability you can say features at the gate level implementation design. So it becomes a problem okay. So designer may not be very much familiar with the gate level logic with this intersizer has synthesized. Because after optimizing typically it is very difficult to identify the gate level logic components and correlative it with the higher level behavior level components okay. So these are the problems. And now let us try to see that what are the different components of costs that are involved in having a BIST implementation okay.

(Refer Slide Time: 07:38)

Costs Involved in BIST

© CET I.I.T. KGP

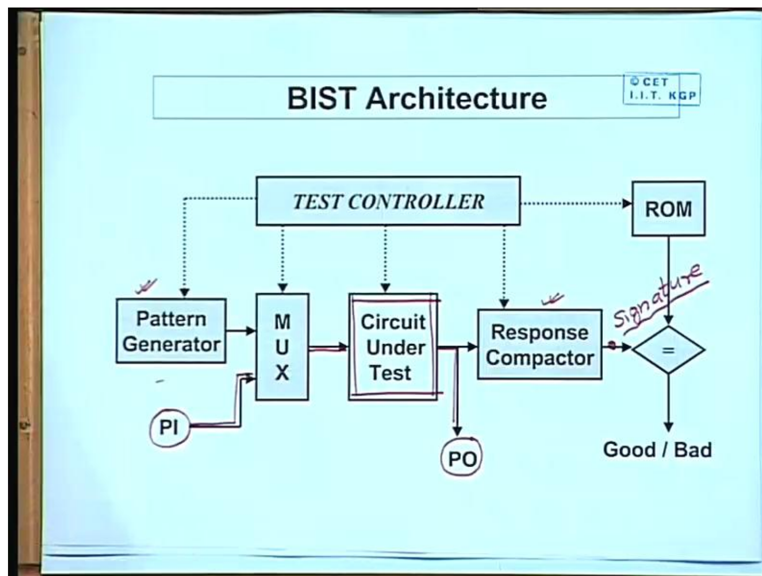
- Chip area overhead for:
  - Test controller
  - Hardware pattern generator
  - Hardware response compacter
- Pin overhead
  - At least 1 pin needed to activate BIST operation
- Performance overhead
  - Extra path delays due to BIST
- *Yield loss*
  - Due to increased chip area

So the first in obvious cost is clearly the chip area overhead for the additional hardware that you need well. If you want to do the testing inside the chip we need two things. We need a hardware pattern generator inside the chip. And we need some sort of a hardware response compacter or response calculator inside the chip. And there has to be a special purpose controller which will

be controlling these two. So these are the three additional hardware blocks that we need inside the chip. And for these we need these extra chip area overheads. Not only that, we need additional pins we must have at least one additional pin which will tell the chip that well now I want to test to activate the BIST operation there can be one pin. Zero means normal operation and one means BIST mode okay. And since we are adding on this extra hardware components there will be performance overhead penalties also.

Because some extra path delays will get in certain due to these BIST hardware that we have inserted and earlier we had mentioned that if we increase the chip area the chip yield will also drop. Now since we are adding some additional components chip area goes up. So yield will also go down. So these are the different components of the cost involved in BIST chip area overhead pin we need one additional pin little pit of performance overhead. And of course associated yield lost. Well of course in today's multimillion device chips this yield and chip area overhead or not that important because the increase will be extremely marginal okay. Now just looking at the overall BIST architecture let us try to selection what are the different because a primary hardware blocks that you need to activate the best and the work just using this.

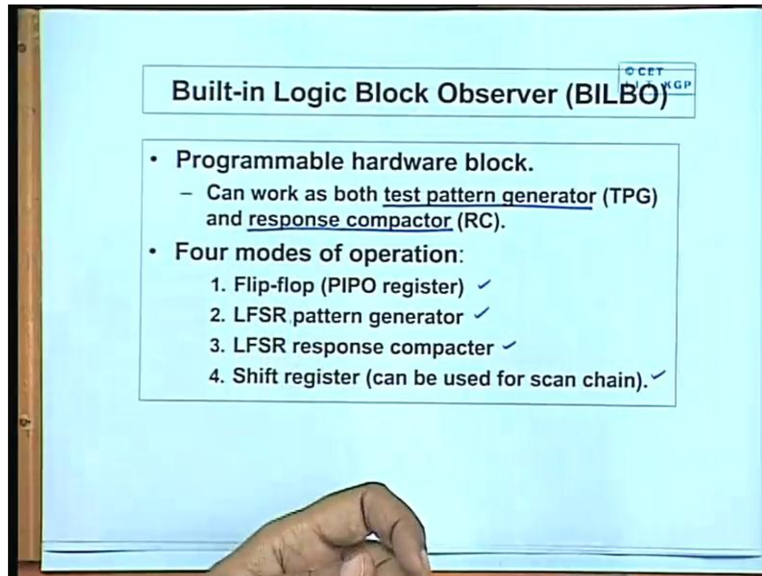
(Refer Slide Time: 09:57)



Well, now in this diagram if you look at, so the circuit that we want to test is residing here at the middle this is the circuit under test. These are the normal primary inputs and these are the normal primary outputs. So this is the normal path of circuit operation. So normally this multiplexer will be selecting this path and this path will get selected. Now you can see that even in the normal mode this multiplexer is coming as an additional delay in the path. So the delay of the total circuit operation will gets slow down a little bit. Now the test mode the test controller will be controlling several things. First is that there will be a hardware pattern generator which will now be starting to generate some patterns that will be apply to the circuit under test. Now the controller will also be controlling the multiplexer so that now this patterns not this PI is will come to the input of the cut CUT. Now since we are trying to evaluate the response also inside the chip.

So it is not practical to store all the responses individual responses and compare because the number of pattern can be as high as say means one million okay. So rather than storing one million outputs and comparing them with the good value stored in a ROM what is normally done is that the circuit responses are compacted. So after compaction we get a very small value out here this is sometime called as signature. So we do not compare the individual responses rather we compare only the signature with the corresponding goods signature that will be stored in a small ROM. So after comparison well we can announce that the circuit is either good or bad okay. So this is the typical BIST architecture. So we will have to talk about this pattern generation and this response compaction in some detail okay. But before going into these, let us first look at some of the popular general purpose hardware blocks which people use in this architecture well. Now if you can easily correlate the architecture with respect to this diagram. Because we need a pattern generator we need a response compactor okay.

(Refer Slide Time: 12:51)



Now there is general purpose hardware block which has been proposed this is called built in logic block observer BILBO. Now BILBO in a sense it is a general purpose logic block general purpose in the sense that well sometimes it can act as a pattern generator sometimes it can act as response compactor. Sometimes it can act as a normal register which is part of the circuit. So instead of instead of investing in totally new hardware what we can do is that some of the existing registers in the circuit we can enhance them with some additional logic so as to make them programmable in the sense.

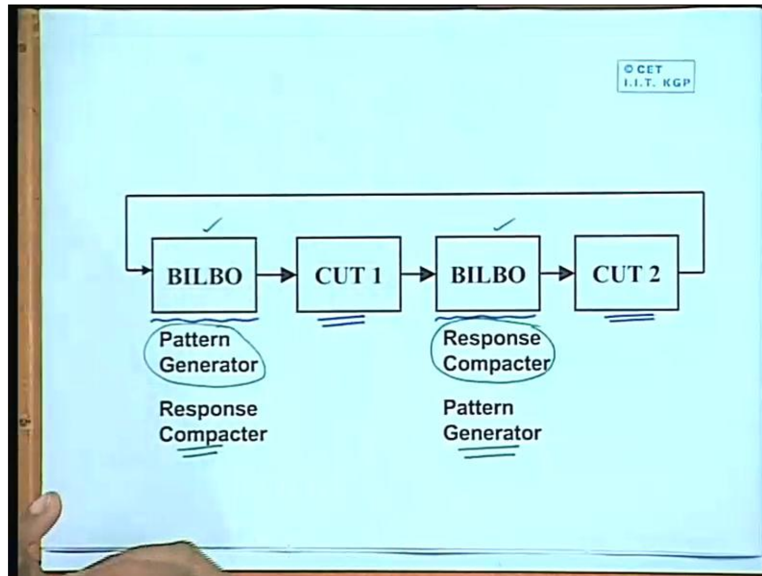
Now we shall see later that we can have very simple implementation of this pattern, pattern generator and response compactor by slightly modifying this circuit diagram of a shift register. So all the since done can be done very easily. But let us first see the capabilities of this BILBO built in logic block observer. So as I said this is a programmable hardware block. Well with respect to testing this has the capability to work as a test pattern generator and also as a response compactor. Now this circuit can work in four different modes. One is a normal flip flop mode it is a parallel in parallel out register.

So in this mode this register can act as a normal register which is part of the circuit. In the second mode it can act as a pattern, pattern generator and in the third mode as a response compactor. Now both these are based on something called lineal feedback shift register that we shall come to very shortly. So this linear feedback shift register is a hardware structure which can be used to generate patterns and also to compact responses and the fourth mode is that you can configure them as a shift register so that if you are using scan path then you can use the shift register as part of the scan chain very easily okay.

Now just one thing I would like to mention here before a proceed for this that we are talking about LFSR based pattern generator or one chip pattern generator. But usually what people do is that the kind of patterns that are generated that I not exactly the type of patterns which are generated by a test generator program which we had discussed earlier. But rather what is done as special hardware circuit is used which can generate very good pseudo random patterns. And I and showed a slide earlier I showed a typical graph earlier that is you feed a circuit with zero random patterns and if there are sufficient number of zero random pattern, then the fault coverage can also be pretty good.

Of course you can assert in this you can verify these through simulation before actually installing an implementing the BIST hardware in place. Before that you can use fault simulation to find out that well actually if I use say for example 10,000 zero random patterns then what is the fault coverage. If you see or find that the fault coverage is not adequate we can add some more patterns okay. So in BIST reducing the number of patterns is not that important. Rather the ease or simplicity with which you can generate the patterns that is more important okay fine. So with the capability of this BILBO let us take a couple of examples.

(Refer Slide Time: 17:01)

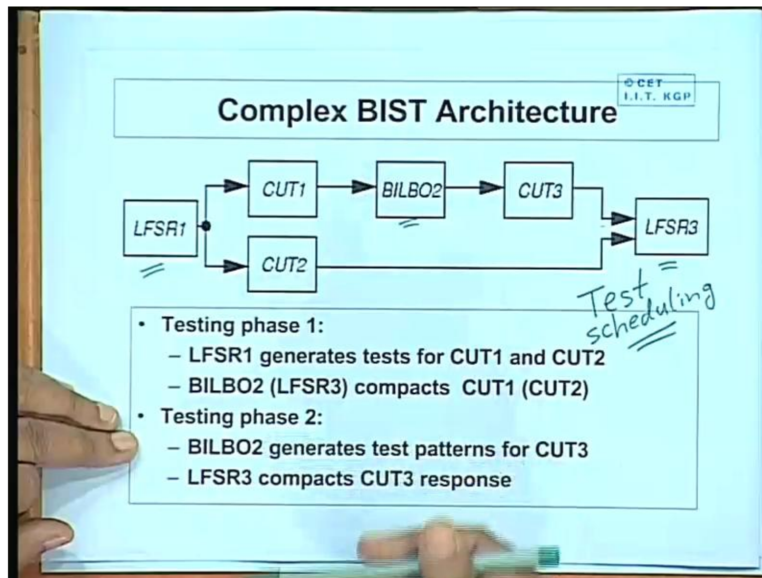


This is a very simple example suppose we have we have some kind of a circuit for there are registers in between suppose this is a register this is a register and there is some combinational logic in between. We call it CUT 1 and CUT 2 and they are connected like this. Now we have to test this hardware. Now we can do it very easily first thing is that we can configure BILBO into a shift register. And by using that shift register we can use the the conventional shift register test which you use for scan path for testing them that 00110011 pattern. So by that we can test the BILBO registers. Then what we can do for testing this CUT? One we can configure BILBO to act as pattern generator we can configure this BILBO through act as response compactor. So now this fellow will be generating the patterns and the output will be response will be compacted here.

So after compaction you can compare with the good value and see that whether it is good or bad. Similarly when you are trying to test circuit under test two then this BILBO will be generating the patterns and this one will be compressive okay. So in this way in a normal circuit by configuring some of the registers as BILBO registers we can make this self-testing work. Of course we need a test controller which will be generating the control signals and the necessary sequencing that first, the first BILBO should be generating pattern this should be compacting. In

a second step the second one should be generating pattern in the first one should be compacting. So these things should be controlled by a separate small test controller we should also the inside the chip okay well. Now let us take another example which is slightly in more complex.

(Refer Slide Time: 19:11)



Let us take an example like this. These are combinational blocks CUT 1, CUT 2 and CUT 3. Well these are some registers say one here one here and one here. Now if you look at the nature of the circuit, this one will be used only for generating patterns never for compaction. Similarly this one will be used only for compaction never for generating pattern because this is the sink. So, only the middle register this one should be configure as a proper BILBO okay. But the other two should not be as complex as a complete BILBO register. They can work specifically either as a pattern generator or as a response compactor okay. So here what you can do is that well while we are testing say CUT 1 and CUT 2, this is the testing phase 1. Then LFSR are one can generate the test patterns these test patterns can be fed concurrently to CUT 1 and CUT 2 we can test both of this together.

Another response of CUT 1 will be compacted here and the response of CUT 2 will be compacted here okay. So testing of CUT 1 and CUT 2 go on together. After that in the testing

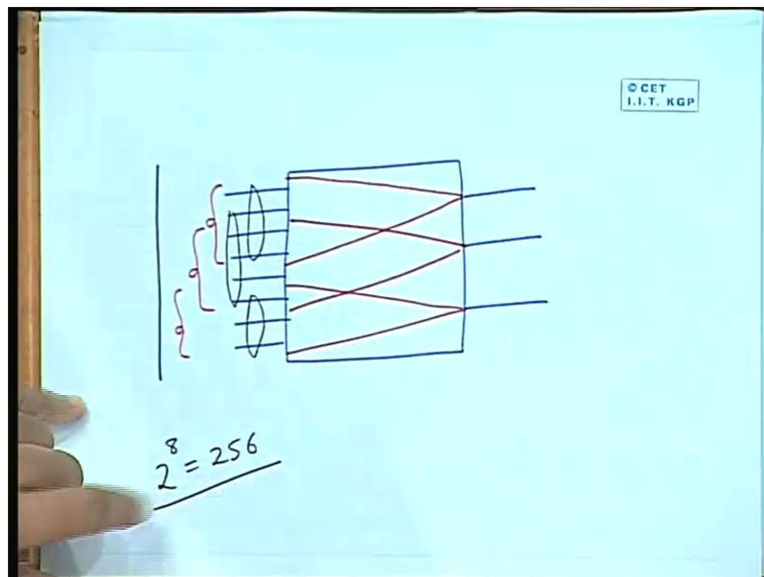
phase two what we can do BILBO two can be generating the patterns then LFSR 3 can be compacting. So that we can test this CUT 3. Now this process is actually called test scheduling. So this is a very small example I have taken. But in general this data path will be fairly complicated and we will have to find out a schedule like this. This schedule for example consists of two phases. So there can be many such phases. So we will have to find out a schedule which will be taking overall the minimum amount of time for testing. So we can find out that in some intermediate step we can we can actually test a number of blocks at the same time. So we can have some parallelism there okay fine. Now talking of the pattern generation.

(Refer Slide Time: 21:27)

Now well we are talking about pattern generation on chip inside the chip. So here we have several alternatives. First is that we can have deterministic patterns which are generated by a conventional test pattern generator and we can stored them in a ROM. But the problem is that the size of the ROM will be very large and this will be an expensive solution okay. Because in general this patterns will not have much relationship among themselves and you cannot use anything less than or ROM to store them okay. The second alternative is that many people have proposed. This method also is that you apply all possible input combinations to the circuit on a test which means the test pattern generator will be a simple counter. But the problem is that if the

number of inputs is for example thirty. Then the number of patterns you need to apply is two to the power 30. Now in a practical circuit number of inputs can be as high as 56 or 100. So then the number of patterns to be applied will become astromically high and it is impractical. So this is too time consuming. Now the third method this also many people have proposed this is something called pseudo exhaustive. Pseudo exhaustive is something like this.

(Refer Slide Time: 23:01)



Say I have a circuit. So my circuit has three outputs there are several inputs okay. Now by analyzing this circuit structures we can find say for example, this output number one is dependent on only these set of inputs. This is called the so called cone of influence the first four the second output, say for example is dependent on these and the last output is says dependent on the last three. So the first output is dependent on these inputs the second output is dependent on these inputs and the third output is dependent on these inputs. Now pseudo exhaustive test pattern what it says is that you try to generate patterns on the input in such a way that with respect to these blocks this block of four this block of five and this block of three.

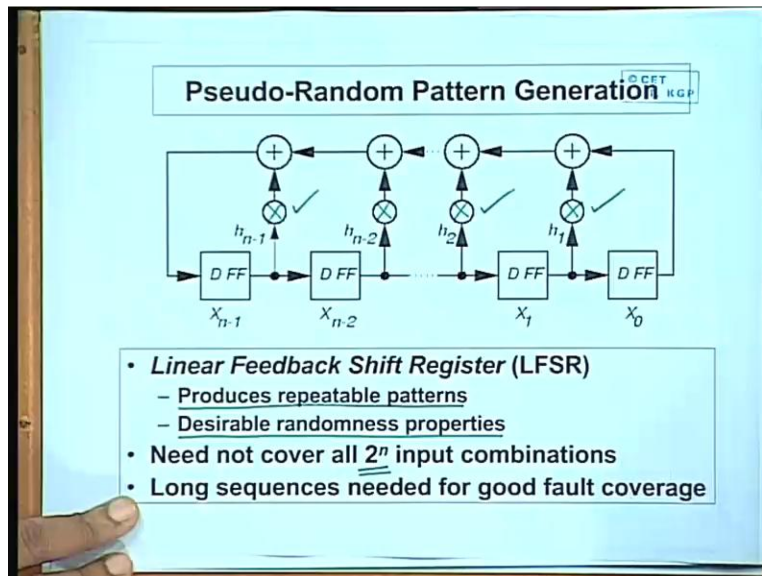
They all get all possible combinations among themselves for example among the first four you get all 16 combinations applied well. There may be some reputation but you do not mind

reputation. But at least all possibilities may be there. Now pseudo exhaustive says that the total input here is 8 for example. So if we apply total exhaustive pattern it would have been 256. But we say that you try to generate a number of patterns which is much less than 256 and which will cover all possibilities within these small subsets. This is the basic concept behind the pseudo exhaustive okay fine. And lastly we have pseudo random patterns.

(Refer Slide Time: 25:12)

Pseudo random pattern in fact this is most widely used because it is the most easily generated one and economical well exhaustive can also easily generated but it is time consuming pseudo random pattern we will see that it is easy to generate and it is also quite practical okay. So now let us look at how we can generate pseudo random patterns using a linear feedback shift register okay fine.

(Refer Slide Time: 25:45)



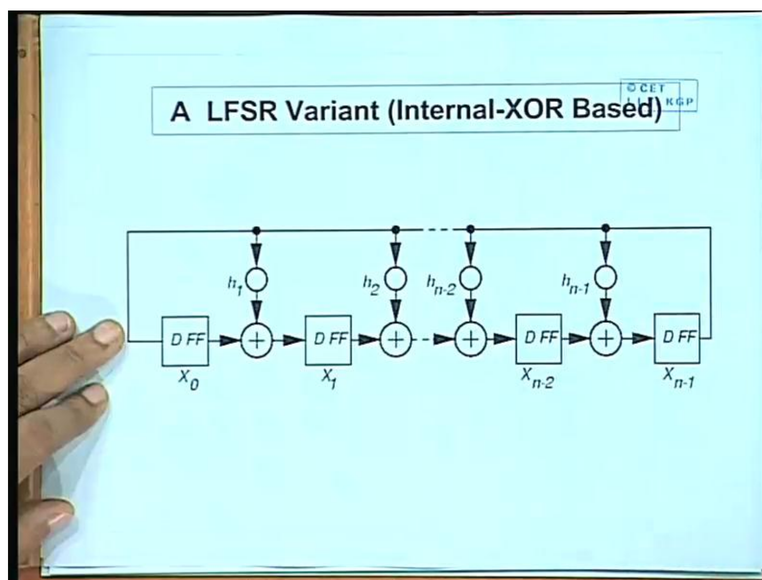
This diagrams shows you how a linear feedback shift register looks like. See if you just ignore this circuit on top of this there are some D flip-flops which are connected as a shift register. So essentially it is it is a shift register with some additional logic. These circles are exclusive OR gates. Well you can ignore these for the timing assumes that they are all connected okay. So the outputs of the D flip flops they are fed to exclusive OR gates this forms some kind of a feedback network. Some combination of the outputs is XOR and the XOR result is fed back to the input of the first flip flop. Now the significance of these circles is that it is not necessarily the case that all the outputs should be XOR. So some of these maybe there say for example these three are there but this is not there.

So you selectively find out which output you want to XOR and you XOR them and feedback at the first flip flop in the input. This is what is called linear feedback shift register it is a shift register there is a feedback network and since exclusive OR is a linear function we call it a linear feedback shift register. Now since it is a well-defined hardware structure, so once you load it with an initial state it will always produce repeatable patterns. And it has been proved through extensive experimentation and analysis that the that the kinds of patterns which are generated by this kind of an LFSR has the desirable randomness properties. So we can treat them as random

patterns. But a true random pattern should not be repeatable since this pattern is repeatable we call it pseudo random okay fine.

Now with respect to testing if it is an n bit register it is not the case that we need to apply all two to the power n input combinations okay. So the number of patterns that you need to apply is typically much lesser for example 10000, 20000, 50000, that should be sufficient. And the number of flip flops can be as high as 32 or 64. But you need sufficiently long sequences for good fault covering but how long that you should evaluated and find out through simulation. Now there are other variances of LFSR also. This is one way in which LFSR can be constructed. And another kind of LFSR configuration is there where the XOR instead of laying output like this, they are inside this shift register like this.

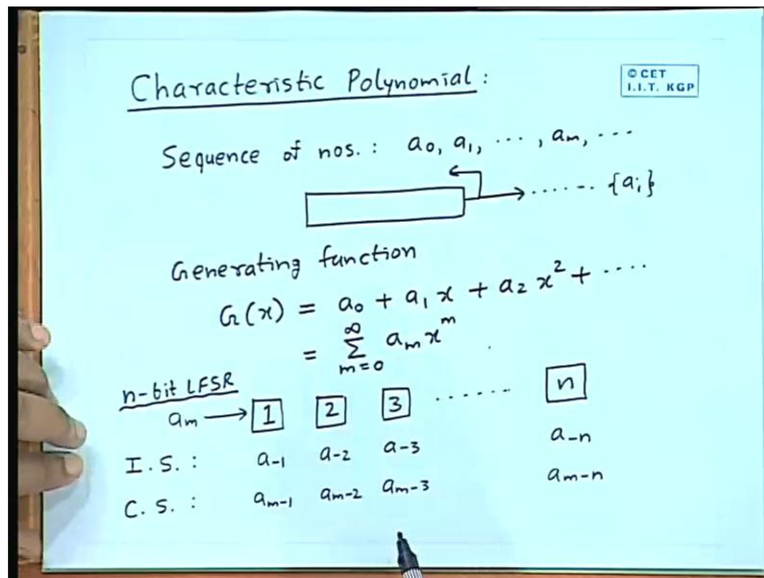
(Refer Slide Time: 28:51)



This is called internal XOR based. So where you have the shift register with the exclusive OR gates in between and the feedback path from the output you have a path to the input. But some of the XOR gates are also fed from here. Depending on the property you want, you will have to select that which XOR gate will have to feed okay. These are the two different variations well. Now let us try to look at some of the mathematical properties of mathematical basis on which

this LFSR based I told you that you can select this feedback points depending on requirement. But actually how do is select that but for that let us first look at some of the mathematical properties of this linear feedback shift register.

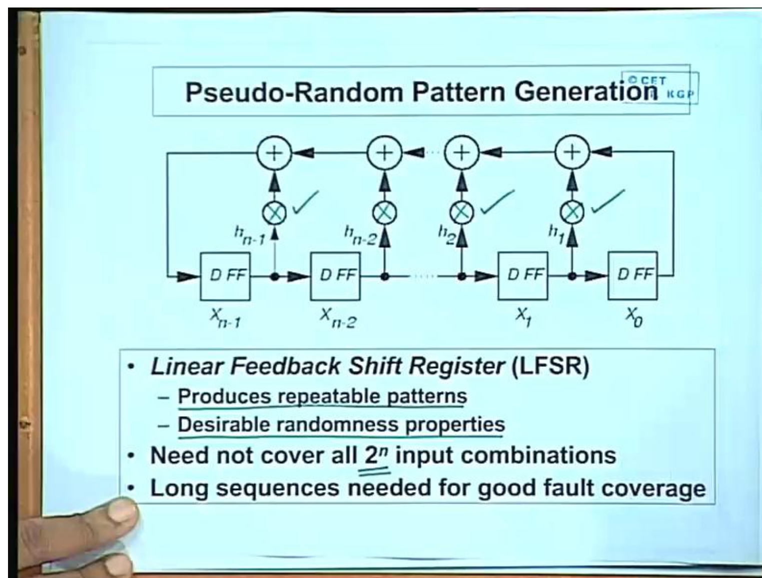
(Refer Slide Time: 29:47)



The first thing we talk about is something called characteristic polynomial. This is something which is associated with an LFSR. Let us see that what this really means well. We will start with a sequence of numbers. Let us call them a_0, a_1, a_m and so on. Now this sequence of numbers with respect to an LFSR it can be treated for example if you have an LFSR like this. So this is the output which is fed back I am not showing the whole diagram. So this output bit this will be generating a streams of zero and once. So this output bit pattern this can be considered as this sequence a_0, a_1, a_2. So this a is ai is or 0 or 1 okay. Now given a sequence of numbers like this a zero to a m we can define or generating functions. So the generating function can be defined as a polynomial in some parameter variable x. So this will be a 0 plus a 1 x plus a 2 x square and so on. So this you can write in closed form as m equal to zero to infinitive, a m x to the power m okay.

Now suppose we have n bit LFSR. You have an n bit LFSR. So let us say these are the corresponding flip flops 1, 2, 3, up to n. Well now a zero a one of the bit patterns, so at the first clock whatever comes out or whatever gets in is a zero. Actually whatever gets out will be fed back here so that is what we call a zero. So we are assuming that the initial state that the initial state of the register is a minus 1 a minus 2 a minus 3 a minus n. And suppose after n clock pulses the current state looks like this. a_m minus 1 m minus 2 m minus 3 and m minus n. And in the current state a value that is being fed back at the input out here this will be a_m. So a_m will be that linear combination of these values okay which will be fed back fine. So now if we just look back at that diagram or LFSR again.

(Refer Slide Time: 33:26)



So here we are set that the outputs of the LFSR they are fed back through some block h_1, h_2, h_n. Now this h is can be 0 or 1 if it is a 0 which means that there is no connection. If it is a one it means that this is connected these are simple multipliers it multipliers with either 0 or 1 okay. So either this is present or not present.

(Refer Slide Time: 09:57)

Characteristic Polynomial:

Sequence of nos.: $a_0, a_1, \cdots, a_m, \cdots$ ...... $\{a_i\}$

Generating function

$$G(x) = a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{m=0}^{\infty} a_m x^m$$

n-bit LFSR

$a_m \rightarrow$ [1] [2] [3] ...... [n]

I.S.: a_{-1} a_{-2} a_{-3} a_{-n}

C.S.: a_{m-1} a_{m-2} a_{m-3} a_{m-n}

So what I can say is that, that this minus one to a minus n these are the states as the present instant. And based on the linear combination of that we are generating this a m. So we can write down this a m as follows.

(Refer Slide Time: 34:07)

$$a_m = \sum_{i=1}^{n} h_i a_{m-i}$$

$$G(x) = \sum_{m=0}^{\infty} a_m x^m = \sum_{m=0}^{\infty} \sum_{i=1}^{n} h_i a_{m-i} x^m = \sum_{i=1}^{n} h_i x^i \cdot \sum_{m=0}^{\infty} a_{m-i} x^{m-i}$$

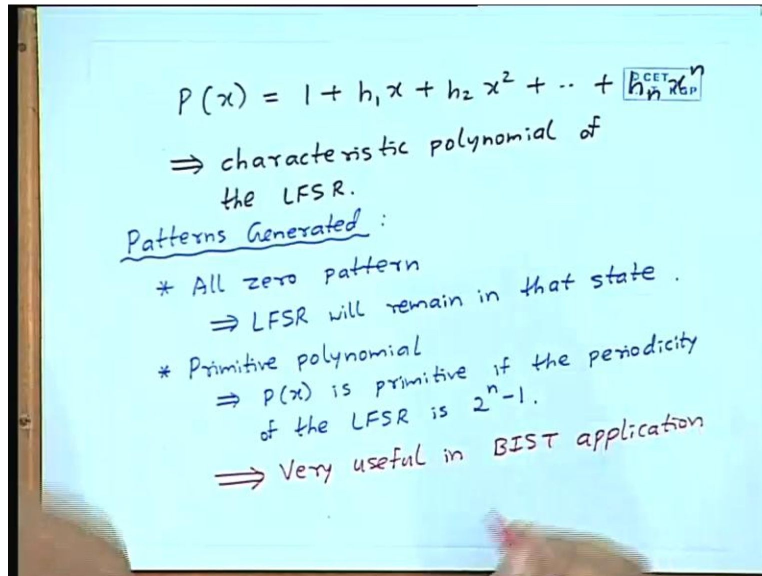
$$= \sum_{i=1}^{n} h_i x^i \cdot \left[\underbrace{a_{-i} x^{-i} + \cdots + a_{-1} x^{-1}}_{\text{Initial state}} + \underbrace{\sum_{m=0}^{\infty} a_m x^m}_{G(x)} \right]$$

$$\therefore G(x) = \frac{\sum_{i=1}^{n} h_i x^i \left(a_{-i} x^{-i} + \cdots + a_{-1} x^{-1}\right)}{\underbrace{1 + \sum_{i=1}^{n} h_i x^i}_{P(x)}}$$

I equal to one to n h_i a m minus i. So if i is one h one a minus one just observe h_1 is the weight (word not clear: 34:28.010) minus 1 h_2 a minus 2 and so on. Well with this a m now again you go back to the expression for G x which we have to computed earlier. So G x we have set is m equal to zero to infinitive a m x to the power m. Now this we can write like this. Instead of a m we can substitute this. I equal to 1 to n h_i a minus i x to the power m. So instead of a m where written down by this expression. Well now we can reorganize this a little bit and we can write i equal to 1 to n we can take out this h_i and xi we can x out of x to the power m i take out xi here. So what remains is m equal to zero to infinitive a m minus i x m minus i this is what remains. So this I can reorganize and write like this because xi and this x to the power m fine. Now this again the first and remains the second term expand this the first few terms are this.

And finally you get m equal to 0 to infinitive a m x to the power m. Now this is nothing but Gx. This is nothing but Gx. And this term this a minus 1 to a minus i this is nothing but the initial state of the LSFR okay. The initial value which has loaded fine. So by reorganizing the equation taking Gx on the side we can and if you we just solve it as little bit you can get the final expression for Gx from here. Gx becomes equal to i equal to 1 to n h_i xi this part a minus 1 x 2 minus one divided by 1 plus i equal to one to n h_i xi. So this is the final expression for the generating function. See if you look at the numerator the numerator is something there is a part which is the initial state of the LFSR. And of course we have a term this hi xi. But the denominator is something which is a polynomial of degree n. So the denominator if you if you expand and the denominator is typically denoted by P x okay. So if you down the expression for P x, now from here.

(Refer Slide Time: 38:15)



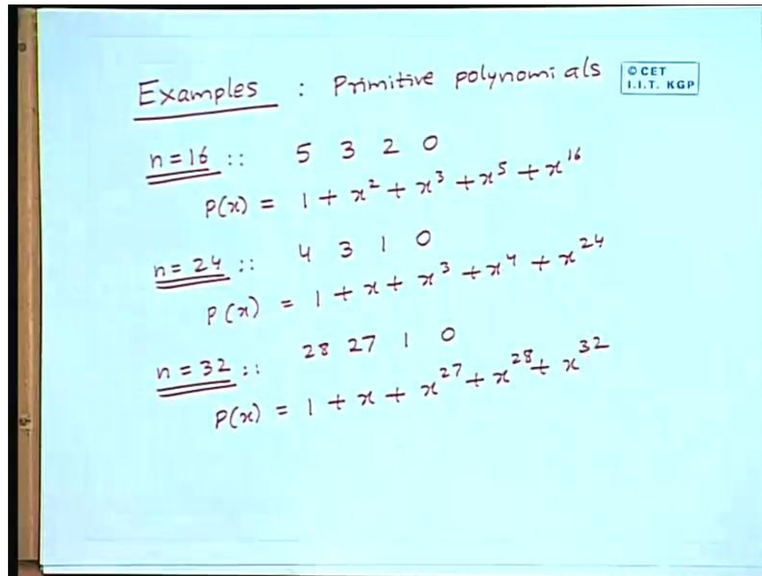
So P x will become 1 plus h 1 x plus h 2 x square plus hn x to the power n. Now this polynomial P x is defined or it is called as the characteristic polynomial of the LFSR. Now it is this characteristic polynomial which defines some properties of LFSR. See if we look at the patterns which are generated. We look at the patterns which are generated by a LSFR. Just one thing you can immediately observe that if you considered the all zero pattern. Well if you load the LFSR with the all zero pattern. Then you can you can see that the LFSR will always remain in the zero state it will never come out of it. Because you are feeding back to XOR function and an XOR function with a set of zeros set the input will give a zero as the output. So it is zero which will again the feedback. So an all zero state will leave the will leave the LFSR indefinitely in the all zero state. So once you have all zero pattern in the LFSR. So LFSR will remain in that state okay. So this all zero pattern you cannot load.

Now there is another kind you can say the characteristic polynomial that we have defined some of this characteristics polynomials defined as primitive. There is a concept of primitive polynomial. The idea is that the characteristic polynomial P x is called primitive. If the periodicity of the LFSR is 2 to the power n minus one. Well what do mean by this? Now we have an n bit LFSR. So an n bit register potentially can be in two to the power n possible states.

Now out of the 2 to the power n possible states we are just set that all zero state is ruled out once it gets into all zero it will remain there. But a primitive polynomial is one in which if you start the LFSR with any non zero state it will go through all two to the power n minus one other patterns and then it will come back. But unlike a counter it will not go through this state sequentially 1, 2, 3, 4, 5. Rather it will go through in a pseudo random order. This is the property of the LFSR.

The successive patterns will be a some randomness properties there will be no two patterns which will be repeated. But yet you can cover all to the power n minus one patterns. This is the very interesting property of a primitive polynomial which you can use in the LFSR. So P x is primitive if the periodicity of the LFSR is 2 to the power 1. So these kind of primitive polynomial is very useful in BIST application. Very useful because suppose you start with a 32 bit LFSR. But depending on actually how many patterns you need you can potentially go up to 2 to the power 32 minus 1 so many patterns. So you have entire flexibility but if you would take an LFSR whose characteristics polynomial is not primitive then possibly you will say you will see that well you can generate only 2 to the power say for example 10 patterns. So after every 1024 patterns the pattern gets repeated. So you will really do not want that you will want LFSR as a pattern generator which can generate pretty large number of patterns as you need okay. So LFSR that realize the primitive polynomial is very desirable in that sense okay.

(Refer Slide Time: 43:41)



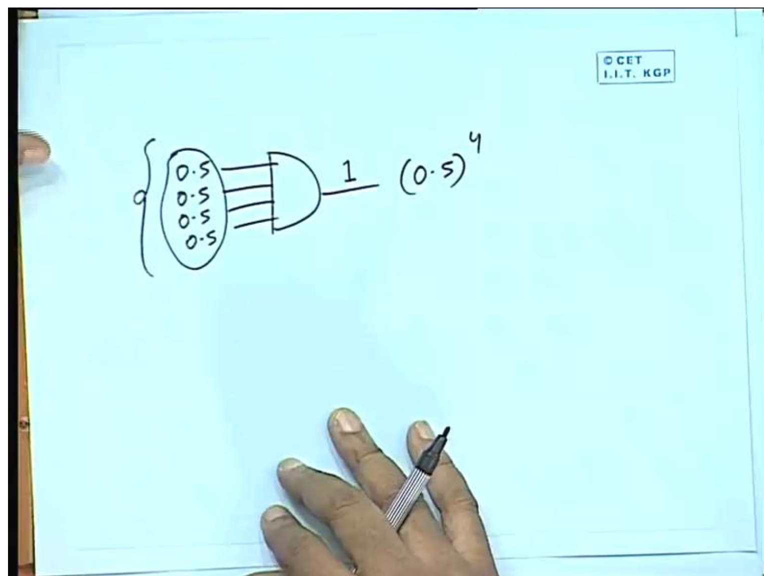
Now if we look at the standard text books for this kind of linear machines LFSR you will find that the primitive polynomials of various orders or listed. They are all methods to find out whether a polynomial is primitive or not I will through exhaustive you can say simulation and experimental studies people are find out a list of all primitive polynomials up to very large values of n also. So I am showing you some I am giving you a few examples this examples of primitive polynomials. Suppose n equal to 16 you have a sixteen stage LFSR. Now in the book we will find its says that the primitive polynomial is 5320.

This actually means that the polynomial characteristics polynomial is 2 to the power 210 to the power 2 x square x to the power 2x to the power 3x cube x five of course the last one will be there x 16 is implied. You take n equal to 24 say for 24 you will see that polynomial like this is listed 4310 which means P x is 1 plus x x cube x 4 x 24. Say n equal to 32, 28, 27, 1 and 0. For this P x is one plus x plus x to the power 27, 28 plus 32. So the idea is that you select the number of bits of the register first how many bits you need. Then you select a primitive polynomial. Once you have the polynomial you construct an LFSR with so many flip flops and you take the feedback point from these.

These are the you can say flip flop numbers from where you need to take the feedback and compute the XOR feedback. So this is fairly simple process. So if you construct an LFSR like this you will get a structure which can generate very good zero random pattern as for your requirements okay. Now I had mentioned earlier that well this LFSR acts as a good pseudo random pattern generator. Now zero random pattern generator a pure pseudo random pattern generator has a property that in each bit position the probability of 0 and 1 or exactly the same. So probability or the signal probability set is “0.5”.

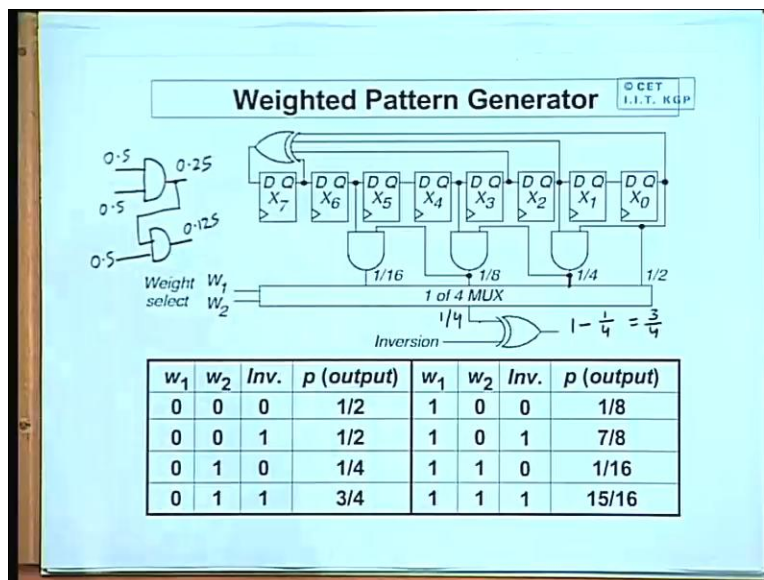
Signal probability is the probability that the line will be having a value of one or since the patterns are random the probabilities of 0 or 1 are equal. Now in many circuit say it has been found and there are many there many works which have been reported in this regard. Their people say that well instead of pure pseudo random patterns if I apply weighted zero random patterns that my fault coverage may improve. So weighted zero random pattern means well you consider and AND gate you take a 4 input AND gate. Now in order to test 4 inputs AND gate say for example I want to test the output stuck at zero fault. So for testing the output stuck at zero faults I have to apply 1 in all the 4 inputs. Now if this signal probability of each line is “0.5”.

(Refer Slide Time: 47:57)



And saying that you have an AND gate this signal probability of each line is “0.5”. So the probability that the output will be one will be “0.5” to the power four this is a very small number okay. Which means that is you apply pure pseudo random patterns the probability that you will get a pattern which will generate a one in the output is very less. So for this purpose possibly you will have to generate a weighted random pattern whose input probabilities are higher than “0.5” right okay.

(Refer Slide Time: 48:38)



w_1	w_2	Inv.	p (output)	w_1	w_2	Inv.	p (output)
0	0	0	1/2	1	0	0	1/8
0	0	1	1/2	1	0	1	7/8
0	1	0	1/4	1	1	0	1/16
0	1	1	3/4	1	1	1	15/16

This can be done very easily. So I have suggested scheme out here. For in the first place you ignore this AND gates we have an LFSR we have a eight stage LFSR. Now this 8 stage LFSR you know that it generates zero random patterns. So you take the output from any bit position. The signal probability will be “0.5”. Now the interesting thing is that suppose I have an AND gate and I feed the AND gate with two inputs the signal probabilities are “0.5” and “0.5”. So what will be signal probability of the output? The output will be 1 is both the inputs are 1. S0 the probability will get multiplied. So the output probability would be “0.25”. So you see that the individual and say you take the output of this here the signal probability is half you take an AND gate. You take any two outputs and it “0.5” “0.5”, so output becomes “0.5”. And you take

another AND gate for one of the inputs you feed with a probability "0.25", the other you feed with "0.5" then what will be the output?

Then "0.5" into "0.25" this will be the signal probability. So it to be "0.125", this is one by 8. Similarly 1 by 8 and "0.5" you put another AND gate you have 1 by 16. So in this way you can have signal probabilities 1 by 16, 1 by 32, 1 by 64 as you wish okay. So this can be a multiplexor depending on this we can select the multiplexor. Now addition whatever you get, for example you just for the sake of discussion. Let us say you get a probability of one fourth here by selecting this line. Now you have an XOR gate in addition. Now if we apply this inversion input as 0. So the same signal will be coming out so you will be get one fourth probability in the output. But if you apply a one in the inversion then this XOR gate will be acting as an inverter. So the output signal probability will now become 1 minus one fourth which is three fourth. So by this simple structure you can get so many difference values of the probability.

So this table list all the possibilities. So you can have this half one fourth by setting inversion to one you can have three fourth. Similarly you can have one eighth by setting inversion to one you can have seven eighth. Similarly one 16 and 15th 16th , so in this way well I have taken a very simple example with three AND gate and one XOR gate you can increase the number of gates you can use other kinds of gates. So you can generate a circuit which can give you a number of different values of signal probabilities are weights. Now using that you can use, you can say, you can use or you can compose or pseudo random pattern generator for the signal probability is can be any arbitrary value well up to certain resolution it depends on. It depends on how many gates you are using more the number of gates high resolutions you will get okay. So this weighted pattern generator in BIST this has been well explored and in fact and in fact many people also suggested use of this.

But in practice we see use this is not that popular well. So we talked about how to generate the test pattern in a BIST where mention that this LFSR is a structure which is very extensively used. And LFSR has some very interesting properties it can generates zero random patterns. If you select if you choose a primitive polynomial, then you can have 2 to the power n minus 1. So many patterns if you need them. So you can excite this circuit inside the chips with this pseudo

random patterns. So the now the next question is that what you do with this circuit outputs that you get. As I said we cannot individually compare each circuit output as it comes. So first will have to reduce them through some kind of a compaction then only we can compare. Now in our next class we shall see that how we can carry out this compaction again using a very similar structure using linear feedback shift register. This we shall be discussing in in our next class. Thank you.