

Embedded Systems-Design Verification and Test
Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture – 36
Part-3: Embedded System Testing
Testing for Re-programmable hardware

Hello students. So, today we have come to the last module of the course on test that is basically we are going to today discuss on Testing of Embedded Software Systems. Till now in the whole phase of the lectures on testing, you might have observed that we are talking of embedded hardware in which case as we have already discussed in the previous lectures that basically in embedded systems we first make a partition of a hardware and software based implementation. And then we have assumed that the hardware part of the implementation is made up of ASIC. By ASIC, I called Application Specific ICs, where we are giving these basically the proper flow that is where we are saying that this is specification, this how to be implemented, and then this is fabricated.

After that, we are dealing with such type of systems, where after the hardware part of the embedded design is freezed, we manufacture it, and then basically we find out test patterns and ways to apply and test those specific hardwares, so that was our assumption till now. But, in the last two basically lectures, I mean we cannot go to a very vast details on these two parts, because these are more or less slightly what do I say newer part on embedded systems or I should say that basically move towards software, hardware hybrid kind of implementation kind of stuff.

So, basically what we are trying to do is that in these two lectures, I will try to give some directions and thoughts. And how basically in some part of the embedded system, which are built on reprogrammable hardware FPGA. In that case, what happens basically, so you download or the hardware is reconfigurable, you can configure it to be an added, you can configure it to be a subtractor, you can configure it to be a multiplier, it can be a processor.

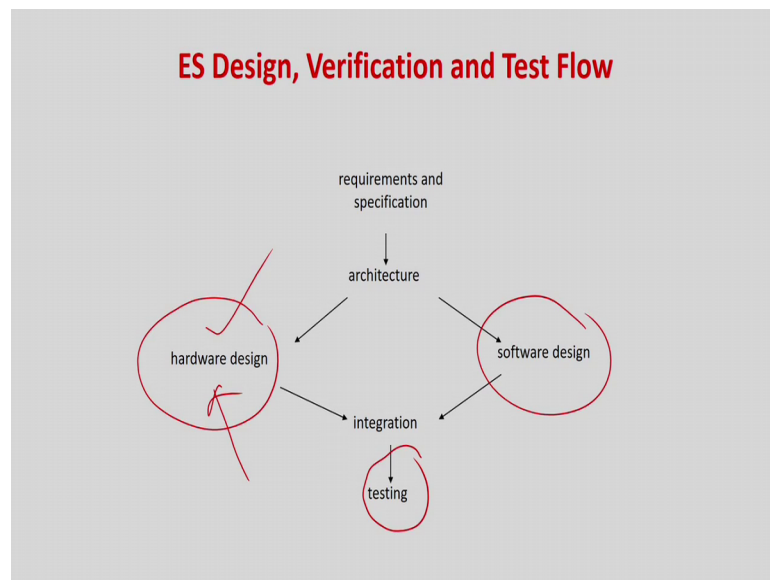
So, basically you have to write the functionalities like a very log code, synthesize it, but instead of fabricating it, you can download and configure that hardware and it will

become the prerequisite or basically the you wanted, you whatever the hardware implementation you wanted, it will become like that that is what is the idea of a field programmable gate array or CPLDs or there are sense of other versions basically. But, as off now FPGA is the most popular implementation of programmable hardware.

Then basically how we should test such hardware or the programs, which has been downloaded over there that is what is the idea of these two lectures. But, again as I am telling you with this is all some of the we can say that the some kind of a bit cornered part of embedded system design and test, whether there I should not call it design, whether the test because in because mainly we are concern when you talk about embedded system test, we rather are more interested or more the literature of the work has been towards exact hardware implementation of which is called hard fabricated chips, we are mainly concentrating on that.

But, a major part of paradigm shift has now been coming by using of reprogrammable hardwares of embedded systems. So, towards the end we felt that we should also give you some thought and directions in which research is going on in this type of platforms.

(Refer Slide Time: 03:12)



So, as we all know, so basically there is a specification. Then the architecture hardware and software as there integration and test basically. So, basically what do we are looking at till now is the hardware design, and we are basically testing it from that perspective.

Like basically there is a hardware, we decided that is part of the form is part will be in the hardware design. And we are manufacturing, and we are testing it that is what is the main thing, and that is what is widely expressed or practice in the testing or embedded system industry. That the idea is see if the hardware design in high is found to be without faults, the software part is verified.

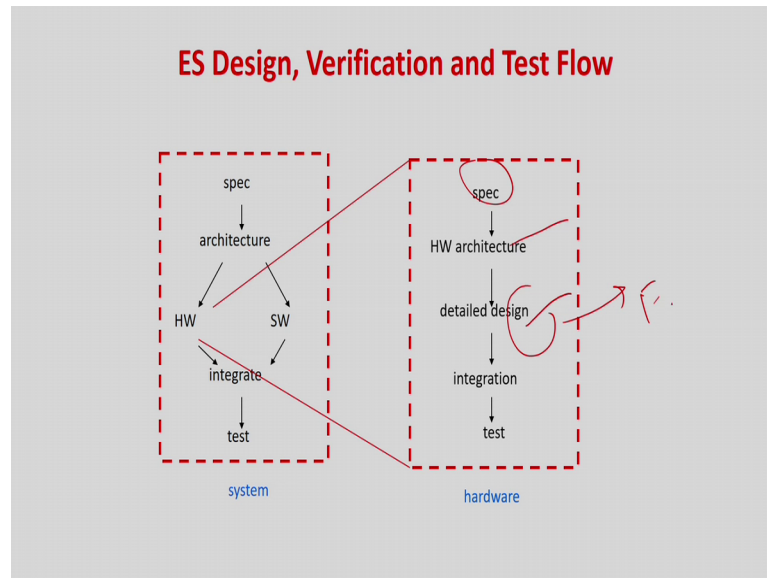
As we have as (Refer Time: 03:44) data is already taught you, then if you integrate it, there should be very less amount of faults possible. And in fact also in the next lecture, we will try to show you that how we can also go for integration testing. But, this is one of the very very complicated ways or complicated subject to understand.

And is a it is also very difficult problem to solve, because hardware is done by one team, software is done by another team, finally they are integrated. Integrated in the sense that the hardware is developed, software is downloaded on that, and software executes on that hardware platform. And you have to find out, if there is any (Refer Time: 04:13) faults due to or errors happening due to integration.

But, again as I told you hardware is again tested be free of fault, software design is verified to be free of faults, then there should not be any kind of integration faults. So, therefore, it is expected that they have should not be kind of any kind of integration error, but still some errors do occur, and people have to find out.

But, this is a more involved topics, who are not directly go into details in this course on this integration testing. But, rather I will try to give you some ideas in the next class that how we can use the software components to test the hardware design or basically the integration that will be the next lecture.

(Refer Slide Time: 04:52)



But, today what we are going to see? Basically, in hardware design is also now becomes some kind of software control. Like a anyway in this is the this is the part that this part is design in hardware, and this part in software. But, if I look at it, so you have a hard specification for the hardware part, hardware architecture, detail design integration, I will test. So, this is the hardware part.

But, see for example if this hardware is also flexible, so what do we mean by flexibility that basically first I thought that I will design adder here. And I will put all the multipliers in the software, because multipliers are more hardware consuming. But, then I after some iterations, I thought that adder plus some more part may be at least 1 2 bit multipliers, I will try to put it over here. At the other part, I will be putting in software.

So, I can try to make lot of trade off that is I keep on changing my ideas. So, before fabrication also I should have a hardware paradigm. In fact, one thing I should tell you that integration testing was not possible till a very till a order of quiet few years back that was not a nice possibility, because you will get the hardware, only when you have your cheap has been fabricating, and it is come to the market.

But, as of now there is something called FPGA-Film Programmable Gate Arrays in which case, you can write your hardware in very long. And you can download it, if you will become your required hardware. Of course, as is a prototype device, it cannot be as

fast as your real ASIC. Because, FPGA as I will you will see is some kind of a general matrix kind of an arrangement. All sort of gates with all sort of inter connects.

So, you can change the inter connects, you can download the level of interconnects you want, and we will be configured to another. You can change the inter connects and the gates, it will become a subtractor and so forth. So, of course we such generality you can implement your hardware or your algorithm is as a hardware, but it will not be as faster as a real ASIC or there will lot of power consumption etcetera.

But, since a very good pro typing environment; so, if you have this hardware design in the initial goals of your design, you can download your software here. And you can start testing, whether the whole protocol system or whole system is a integration will test or not. So, nowadays this FPGA base design is a lot being used in prototype testing or prototype, co validation, co testing, because now you at least have a hardware before the finals applicator cheap is there. You can download the software, and you can see whether it is operating in integration.

Of course, at speed testing, we will not be valid cannot be tested over there, because speeds of FPGAs are not cannot be of course cannot match with a very high speed ASIC implementation. But, of course whole functionality can be verified, and nowadays very large capacity FPGAs are available in which a processor can also be downloaded, and it can be emulated.

So, basically size is not on a concern. So, with this new kind of designs are coming up. So, if for some cases, your speed is not very high speed, because I should not, so that FPGAs are very slow. But, gigahertz level FPGAs may also be available in the market, but as which are called rocket IOS based FPGAs.

But, basically still I mean first many of the applications we do not require a very very fast ASIC device. So, in FPGA is more or less capable of handling this pin. In that case, why I should go for ASIC fabrication, which is a more expensive procedure. So, what I can do, I can take a FPGA with rocket and high speed IO, download my code there. And I can say, this is my ASIC this is my implementation instead of going for a fabrication.

So, nowadays many people are have started using FPGAs not only at the prototyping device also in the final design, so that has brought lot off challenges in the embedded

system market. So, if it is a prototyping device, I am not much bothered about the hardware correctness or hardware faults in the FPGA.

So, if some faults are found in the FPGA or in the code, there will be bugs and then you can debug basically. So, if you find out that because prototyping stage, so your more or less analyzing your implementation, so you can find out that ok. Your software design is fine, if hardware is also fine, but there is a manufacturing faults in the FPGA. So, it is not operating file.

So, in that context, it is important to test FPGAs, but not of a major concerned as in the case of ASIC base testing. But, with the philosophy of now coming into picture, where people are using FPGAs not only the prototyping device, but also the final product has making the test challenge much much higher that is there very large size of FPGAs. 10 years down the line FPGA capacities were very less, you can put only small modules and do the emulation, and prototype test.

But, nowadays FPGA capacities are very very large. So, you can download even processors or in many cases you can put multiple FPGAs to implementing a large, very very large embedded system on the or a set of FPGAs, which are download in a board which are fabricated and arranged in a perpended circuit board. But, an of course the speed is should be slightly lower or the application speed of those application should be slightly less, then such solutions exist in the market.

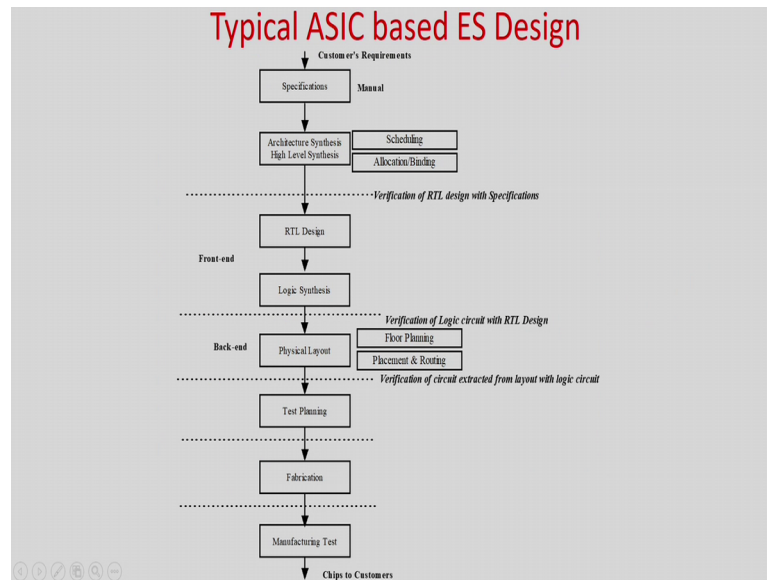
But, now if for such applications people are not at all going for ASIC fabrication, they taking FPGA downloading the code verifying everything is fine they will make is the product, and cell it in the market. So, FPGA testing has become very very important in that perspective.

So, why do I calling as a software, because if you look at the topic of this course, it is testing for reprogrammable hardware in the context of embedded software systems that is in this case, I write a embedded software, I write a hardware. And I do want go for fabrication, whether in the sound version and download in FPGA, and basically software becomes a hardware.

So, there is lot of flexibility or it is a more of I can say hardware software hybrid kind of a nature. So, we should that is why this lecture is focusing on testing of FPGAs, which is

some kind of a hybrid of embedded hardware and embedded software right. So, basically in this pure software is the pure hardware, now in the detailed design basically now instead of an ASIC, it is an FPGA. So, what is FPGA etcetera, we will first see before we go to the details.

(Refer Slide Time: 10:09)



So, this was our if you look at this is standard ASIC design flow basically in which is your specification, then we RTL design, then we can something called logic synthesis. Logic synthesis means, all your RTL should be translated into logic gates, then there is a physical layout, and find is a fabrication, test plants of course coming to the between. So, this is basically proper ASIC flow in which case you give a design, you translating into gates laid out, and finally it is fabricated according to your layout. So, once this layout is being done, you cannot change anything it is the hard design.

(Refer Slide Time: 10:40)

FPGA Principles

- A Field-Programmable Gate Array (FPGA) is an integrated circuit that can be configured by the user to emulate any digital circuit as long as there are enough resources
- An FPGA can be seen as an array of Configurable Logic Blocks (CLBs) connected through programmable interconnect (Switch Boxes)

Now, basically as you will see if I do something like instead of going for a physical layout, instead if I could have got, something like a I can see a n cross n matrix, like something like this. May be I have a 4 cross 4 matrix. And if I give this choice then any of the body can connect to any of the blocks, and each of the block can be AND gate OR gate NOT gate any gate, it can be configured.

There is it become something like a universal logic implementation or 4 gate universal logic implementation. Any gate can be converted to any gate, and they can be (Refer Time: 11:17) own fashion. So, you can implement any logic circuit with these 4 gates. So, it is a something kind of a erasable, and so erasable. Once you download, this may become AND gate, this may become OR gate and so forth. And the interconnection is done. You check your implementation, then you switch the FPGA off everything will be clear.

Then you can again download another set of gates, and make a another type of connection. If something kind of a white board, you can connect any type of gates you can do any type of connections make it work, and then you can you can erase it that is the beauty of FPGA. So, basically the idea of FPGA is a integrated circuit that can be configured by the user to emulate any digital circuit as long as there are enough resources that means, of course if it is a very very multi code process that cannot be done

in a single FPGA, but as I told you. Nowadays FPGA capacities are very large, if in big procedure course are implemented in multiple FPGAs in a board level solution.

So, nowadays FPGAs are very large capacities, so that is why not only it is use for prototyping is also use the final product. But, anyway FPGA is basically in the main logic behind this FPGA came into the market was basically as an emulation platform. You can take a hardware downloading it, sees working erase it, put another hardware.

So, FPGAs can be seen as an array of configurable logic block means, it can be configure to any type of gates. And modern FPGAs, there are lot of additional functionalities, which can be configured like you can have a simple two bit adder, some memory blocks, etcetera can be there. But, any way in this lecture we will be limiting as to simple traditional FPGAs called some old FPGAs in which case, basically we have it can be converted into any type of logic function, there can be of it can be converted into a sequential function and so forth.

So, basically FPGA can be seen as an array of configurable logic blocks, you can convert to any type of gates, and with interconnect which are programmable, because you can connect from any gate to any gate. Because, in ASIC you have tell that this is the AND gate, this is how you should be kind of the OR gate. But, in case of FPGA there is a full flexibility of n cross n connectivity is there, you can connect any gate to any gate by changing the fuses, so that gate 1 gets connected to gate 2, in the next iteration gate 2 maybe connected to gate 1 and so forth by changing the matrix of connections.

And any block can be converted into AND gate, OR gate in the iteration. So, it is a full-fledged reusable programmable hardware. So, you download it is a bit map or I call the connectivity and CLB configurations, it becomes one circuit erase it, do another in download, another you will be a another circuit and so forth. So, mainly so that is why, I called it software based embedded system application. You write code, download into the FPGA, it will be get converted.

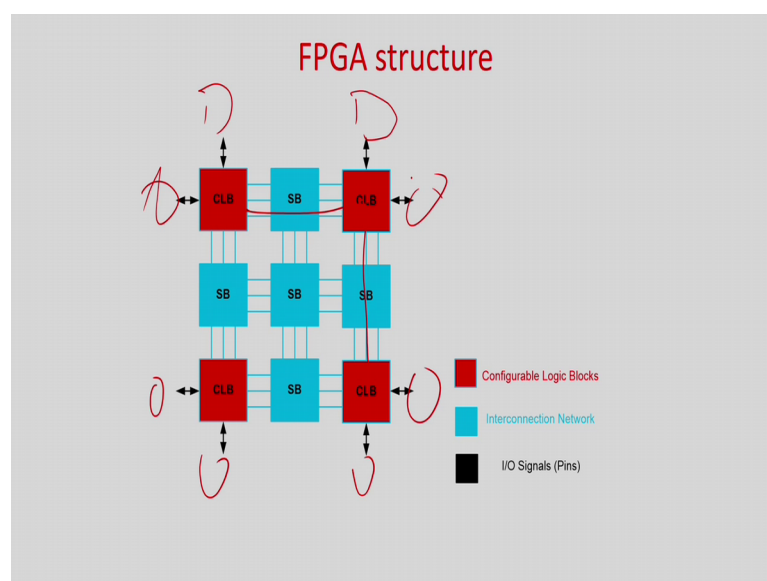
Nowadays run time (Refer Time: 13:47) hardware are also available. This up to this point of iteration or implementation this was a AND gate. After some time, it is reconfigured, it becomes an OR gate. So, run time when the embedded system is running on the hardware, still the hardware can be reconfigured. So, by again which is a very very recent, they will and in case of a FPGAs.

But, the one we are discussing in the lectures on non-reconfigurable run time that means, before the you have to erase the FPGA, download you require required configuration, it will become a circuit, then it will operate throughout its life. But, again if you switch it off, it will get erase. Then you have to again download the new code a frequent, it will become another circuit, and you have to go about it. So, it is not a run time reconfigurable that is what was a traditional FPGA.

But, nowadays it is more powerful, you can do this configuration run time. So, you can understand the power of FPGAs. Nowadays as I told you, it is become a very very high capacity FPGAs are there speeds are also becoming higher. So, many real life application instead of being instead of using as a prototyping device people are downloading the original application, and selling it to the market as an FPGA is called to save the cost of product development and resources.

Basically, because fabrication is an expensive procedure; if some error happens in the design, you cannot do anything basically. If some you made a design, you verified it, still you find out that some there are some errors somewhere, you fabricate it, the whole product whole range of chips you have fabricated has to be thrown out, it is a very expensive procedure. But, if you do it in a FPGA, it is very very flexible. So, there is a FPGA is becoming a very very prominent hardware in the embedded system market.

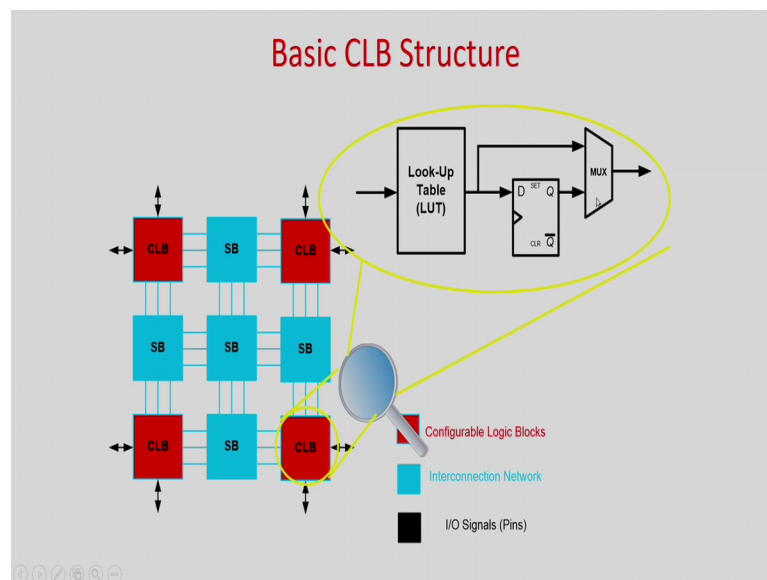
(Refer Slide Time: 15:13)



So, without telling more stories about the FPGA, let me tell you what it looks like. So, here something called as CLB, then there is this is the interconnect, because these are interconnect means, if I want to connect this CLB to this CLB, it will be routed like this. For example, if I want to connect this CLB to this CLB, I will be connecting like this.

So, the blue blocks in FPGA can be switched to make the connections. So, for another application, I may be liking to connect the first CLB to this one. So, SB this SB, SB will be responsible for doing, it is a switching matrix. And of course, the black lines are basically nothing the IO pins. So, there are this is some IO blocks will be available over here, which will connect the FPGA to the outer world right.

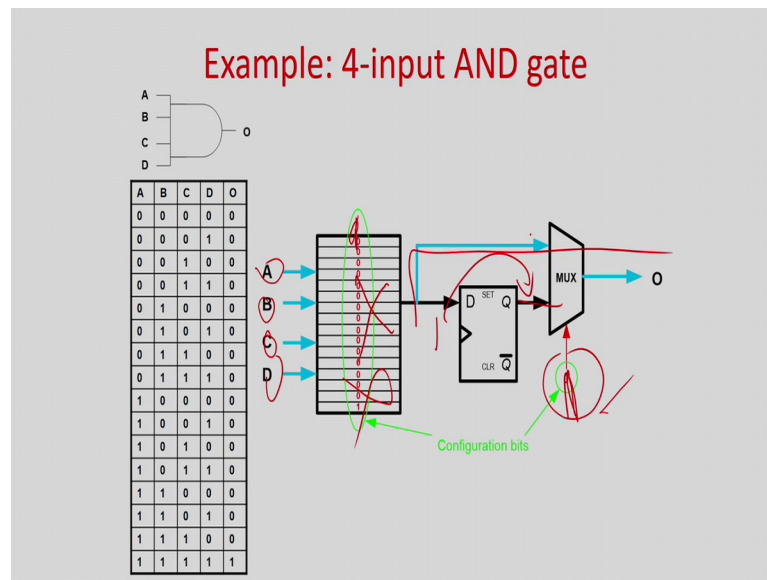
(Refer Slide Time: 15:54)



So, these are very simple architecture one figure will be clear, let me just zoom it. So, CLB is nothing but a very simple stuff like this. Of course, as I told you again a we are talking about the very basic FPGA, moral FPGAs are more sophisticated CLBs. So, there are look up table, the look up table is nothing but it will implement a logics function.

So, it can directly go as the output or it can feed a D flip flop. So, if this feeds are D flip flop, it becomes a sequential element. And if you want to feed it directly to the MUX, it becomes a simple gate. And also I can implement it as a simple MUX flip flop also. In that case basically, the MUX will send this value as the output right.

(Refer Slide Time: 16:28)



Now, again as an example, always an examples are better. So, basically let me say that I want to implement a 4-input AND gate. So, this LUT is nothing but a memory as simple as that because, I as you know the memory is a universal logic gate. I as you all know you might have if you have forgot please go and, it will digital design lectures. So, like for example, in this case because a memory is a nothing but truth table. Like in this case if you look it is a 4-input AND gate, I can have any other implementation. So, it is all zeros will be there accepting ones, because 4-input AND gate. Now, what is this I can think that this LUT is nothing but the memory.

So, in this case this is the address bus 0 0 0 0, so this one will be address. Let me zoom it for you very simple, so all 0s means, it is first block. All 0 0 0 0 one this, 1 1 1 1 means a last element of the LUT will be axis. So, if you see is nothing but this whole truth table is implemented into this memory with this as the address box as simple as that.

So, LUT is nothing but whatever gate or whatever logic, you want to implement. Write the truth table, and convert it into a memory, and that is done. So, basically this is the address bus and this is the memory. So, it is just by the address lines, you can put it up. So, whatever gate this is an and 2-input AND gate, it will be something like 0 0 0 0 1. So, the inputs the inputs lines will be A B 0 0 0 0 0 0 0 0 1 1 0 1 1, so this will be the memory for the AND gate. So, 2-input AND gate.

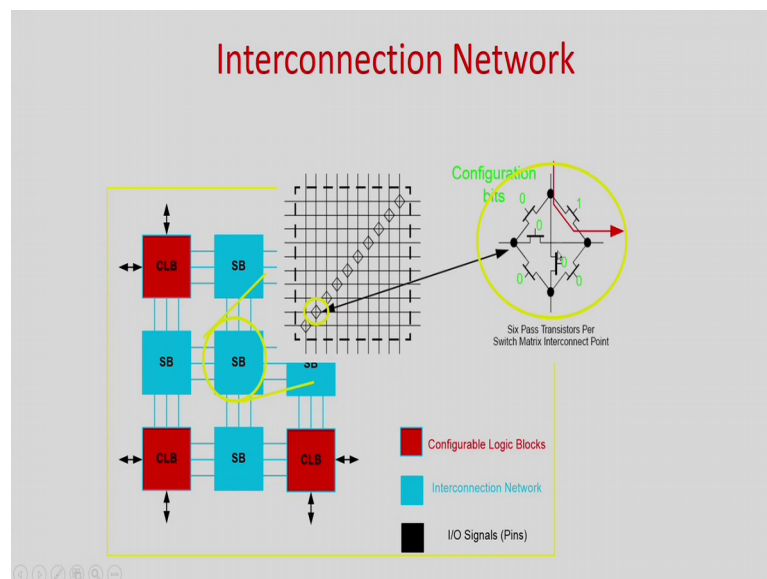
It is a OR gate basically, it will be 0 1 1 1. So, if both A and B are 0 0, this is 0 for all other cases the answer will be a 1. So, whatever logic function you want to implement, you have to write this over here correct. Now, basically it has two outputs, so the output will be D flip flop or you can basically give, it the MUX.

So, if it is want just input of 4-input AND gate, so the configuration bit is zero. So, this is feeding as the output. And if you want to implement as a flip flop, it is very simple, you have to make this as a 1 right. And maybe whatever value you require, basically you have to configure this two all 0s. If you want to use as the single flip flop, and whatever value you want to axis has to be return in this sense, all others basically do not cares.

So, it is something like you make the configuration 1, so this one will be fed over here. And all the inputs you make 0 0, so that you can axes the first cell. You can also be the last cell, depending upon your FPGA implementation. Then whatever value you keep over here will be here if you apply a clock, it one will be coming over here.

So, with this configuration, it will become a flip flop or with the simple configuration, it will become a simple logic function implementation as simple as that. So, this is basically a CLB. So, I think it is very clear to you that CLB is nothing but or it is a gate or some functional logic functional implementation or a simple flip flop.

(Refer Slide Time: 19:02)



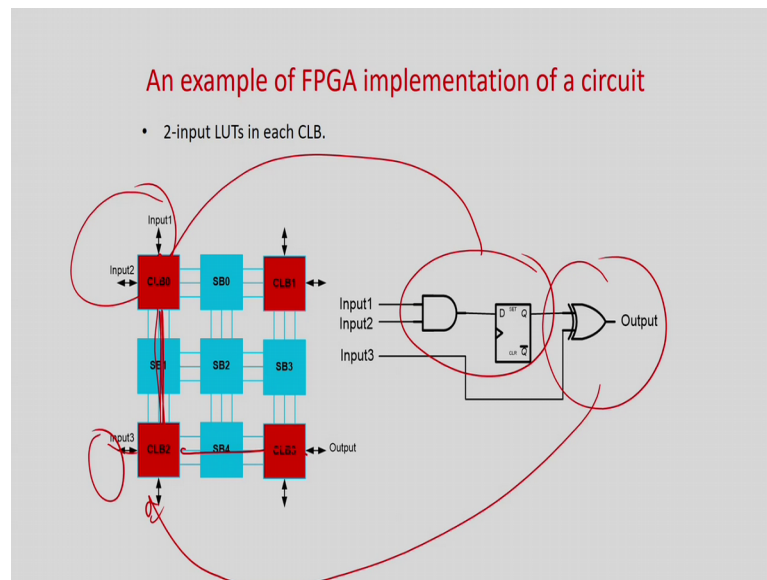
Now, this interconnection is very very important, because you should be able to connect in four directions. So, if something like this, if we look it is pass transistors based logic. So, in this case if you see if you if this part has to be corrected to this, then basically this transition has to be made 1. It is a 1, so it 1 will be connect here.

In this case, the direction is a four line. So, it can be going to any directions. So, if I make this as a one as you can see, this is how the line is connected. If I want to make a connection from this to this, so I have to make this bit as a 1, so this 1 will be connection may. Here want to make a connection like this, it 1 has to be a 1.

Similarly, if this line has to come over there, you have to make it as a 1. But, if these two lines has to be connected, it has to be made as a one. If you want to connect this two lines, basically you have to make it has a one. Of course, you should check that there should not be any kind of shorts so, as again the configuration if you in your paradigm, you can select accordingly.

So, this is basically nothing but a pass transistor logic, which implements that any of these four directions can be connected to any directions as require. So, now if you want to implement a circuit, what you have to do? It is very simple, first we have to configure the different CLBs by the values that the LUTs. And what is the value of the MUX input, so that we will convert all the CLBs to basically your desired logic gates. And the connectivity, you can very simply implement by basically change in the switch connection.

(Refer Slide Time: 20:16)



Again without telling stories take an example. This is a very very simple circuit, and I want to implement it. So, what I require to implement AND gate, a XOR gate, and a flip flop, so I will try to do that. So, any way first you have to do a mapping, so I there actually we are having a flip flop, and this connectivity. So, there are some as you see that LUT can be having again there are flip flop together it can be separate the flip flop or it can be another simple gate. So, in this case it is a gate connected to a flip flop, so you can directly take it over here.

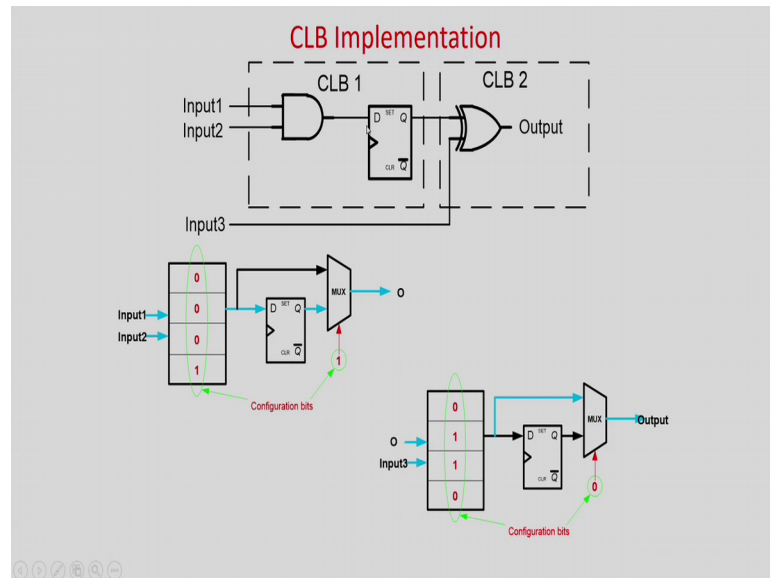
Generally, the idea is another told you, generally you do not have a single flip flop. So, this situation as I told you that making all these as a excess making at 0, and this the 1 or 0, whatever required you can bring it here with clock rarely arises. Because, you do not have generally a single flip flop as a implementation or flip flop as a combinational cloud behind this. So, this is the combinational cloud.

So, in this case the combinational cloud is basically a AND gate. So, you mapping to CLB 2, again this is CLB 0, this is an XOR gate, you have to mapping to some other CLB. In this case, they are mapped it to CLB 2. And basically the CLB 3 is going to be the output basically or because there is or basically, because 3 inputs are there, these are 2 inputs over here, and there is 1 input over here.

And basically you can take it as the you can take it as the output over here or you can take it the output from other CLB that depends upon your necessity. So, basically in this

case 2 inputs for this one. In this case, if you look at it, so one the output of this first block this one will be should be coming over here. This is the input number 3. And this is basically map to the XOR gate, and you can get the you can easily get the output from here or for some reason you can even bring take this output to other CLB depending on that. In this case for some reason, they have taken the CLB output over here.

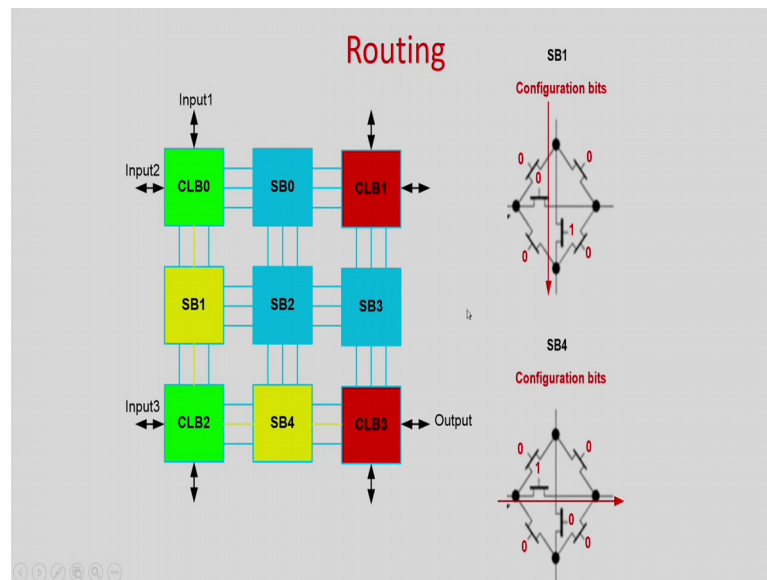
(Refer Slide Time: 21:56)



So, now how do I configure, so simple. So, let us look at the CLB 1. So, you have to convert this one into an AND gate, so 0 0 0 0 1, and basically this is the case. So, let me zoom it for you. So, in this is a AND gate, of course all 1 1 only there answer will be a 1, you have to make it as a one, because basically it should become into the flop.

So, very simple this is CLB 1 as simple as a XOR gate. So, XOR gate means, it is what 0 0 0 1 and 1 0 1 and this is actually 1 1, so 0. So, it is a simple gate. But, in this case the flop the MUX has to be made 0, because I do not require a flip flop over here, basically this should be the output. So, as see very simply, I can implement this two basically is in two CLBs, I can make the implementation of the circuit.

(Refer Slide Time: 22:37)



Now, routing. So, as I told you the first implementation input one will go over here to this connection, so this is what is being implemented over here. Basically, this connection this connection I am implemented. So, this is what is the case. Now, secondly basically means input two is already there. And this one, I am as I told you somehow I am going to take the output over here. So, basically SB4, it should make a connection like this. I could have also taken the output over here, but anyway to show that two routing tables we are two routing we are using. So, anyway the example, I have taken from the general this is where general textbook material available in most of the text books on VLSI design.

So, in this case they have routed it over here. But, if you are if you want to take the output over here, then this is not frequent, then this is also not frequent. But, if you want take the output over here, so SB4 has to be made in this manner, so that basically this is as a 1, so the output is so this connection is done. And basically this configuration basically is your implementation of the simple circuit like this. Now, what I have to do, so you know that this is the configuration, which allows to do.

(Refer Slide Time: 23:39)

Configuration Bitstream

- The configuration bitstream must include ALL CLBs and SBs, even unused ones
- CLB0: 00011
- CLB1: 01100
- CLB2: XXXXX
- CLB3: ?????
- SB0: 000000
- SB1: 000010
- SB2: 000000
- SB3: 000000
- SB4: 000001

Now, what happens? Now, given an FPGA, you have to configure all these CLBs and SBs you have to download the value. So, what I will do? In this case, important are CLB 0, CLB 2, CLB 3 maybe an SB1 an SB4, but all have to be downloaded, because you cannot keep any of this CLBs empty. If they do not require, they can be x x x I do not believe, do not understand, whatever you want to download you can download, but that is not of a concern. But, important is whatever I am requiring for my implementation.

So, if you will see CLB 0 means, 0 0 0 0 0 1 that is the content, and it has throughout as a 1. What is that, so it is this one is the configuration, and this one is the bit. So, it is the input basically to the LUT, and MUX MUX is a 1. So, they have mirror configuration like this. CLB 1 is basically if you look it is an XOR gate, so it was 0 0 0 1 1 0 that is the XOR gate. And this was a 0, because the configuration was something like this, because I want to directly take as the output without a flip flop, so I will write like this.

Other flip flops I do not bother, basically then important is basically your SB1. So, SB 1 is something like this which should be a 1. So, basically similarly in case of a SB1, you say this is a 1. These things I do not bother look at much. SB2, SB3 all 0, they have kept means everything is disconnected they have disconnected everything, and SB4 also should be something like this. So, only one bit is a 1. So, in that case this is SB4. So, this is how the configuration SBs done.

So, all zeros means every it is open, so it is not connected to anyone, this also not connected to any one. You can download you can some make some arbitrary connection or sometimes you can keep it and open. So, in this case they have kept it an open. So, what I do? So, this is what is something called a bit map.

So, there are automated tools you tell the design is a design is I want to implement, it will automatically map your designs, first it will synthesize a design, it will map into the exceptive CLBs. And basically it will find out what switches to be made on, and what switches to be made off, so that proper routing is done. And then there will generate something called is the bit map, which will be download to the FPGA, and basically if FPGA will become your hardware that is what is actually call field programmable gate array.

So, why it is a we tell it is a software base implementation of a hardware, basically because in a embedded system, we first partition hardware and software. Hardware was manufactured ASIC, so you cannot do much changes, software is downloading the hardware and tested or use in the community. But, now we call it hardware software kind of a hybrid kind of an implementation, because it is a I have not fabricated an hardware, I have written a very log code and hardware language code. Instead of making a hardware, I am downloading it one FPGA, it becomes hardware, which implements the software, but it is a soft implementation.

Some of the modern FPGAs, you can configure all this six in the run time, which part of the circuit is not doing its word, it can be reconfigure that is a very modern concept. So, if any of the hardware can be reconfigure run time, so it becomes some kind of a hardware software hybrid implementation or hybrid level of thought also basically.

Hybrid implementation initially was thought has hardware is fixed, software changes. But, here the hardware software both changes, so that is why FPGAs brought into a paradigm shift in embedded shift implementation. But, the course expression we are not dealing with such high and FPGAs in this course, our case everything is predetermined you have to just download this code, it will become your hard so hardware. So, this is lot of flexibility.

Nowadays, I mean you need not partition your I mean hardware software partitioning so basically drastically, because I mean no changes can be done after that, right now with

FPGAs that assumption is no longer valid. After some time, you will find that some more part could have been go into hardware or you can change the paradigm it is possible, because FPGA is very very flexible, so that is why in this course, we are trying to look at how FPGAs are also getting tested.

(Refer Slide Time: 27:14)

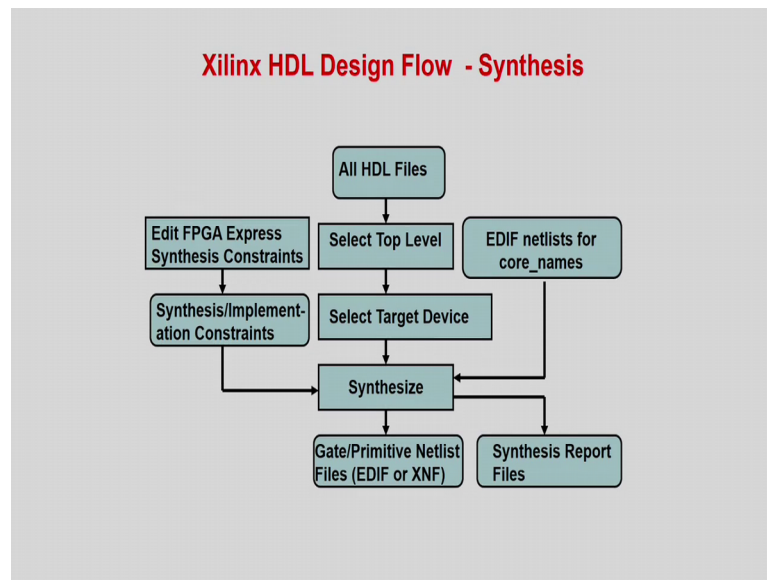
FPGA EDA Tools

- Must provide a design environment based on digital design concepts and components (gates, flip-flops, MUXs, etc.)
- Must hide the complexities of placement, routing and bitstream generation from the user. Manual placement, routing and bitstream generation is infeasible for practical FPGA array sizes and circuit complexities.

So, any way this tells you that what is an FPGA. So, of course we cannot do that manually though there should be FPGA EDA tools, which will provide a design environment in which case, it will hide all complicity like placement, routing, bit stream generation etcetera. We just give the circuit, it will automatically map it to the say CLB respective CLBs, it will make the automatic routing, it will generate the bit map, it will be automatically download, and you will get the circuit implementation.

The lot of FPGA tools like Xilinx, mentor graphics, Altera. There are many lot of vendors, I thing you many of you are may be knowing, who are into this FPGA business. So, there it is very simple you write the code synthesize it, after everything is automated.

(Refer Slide Time: 27:52)

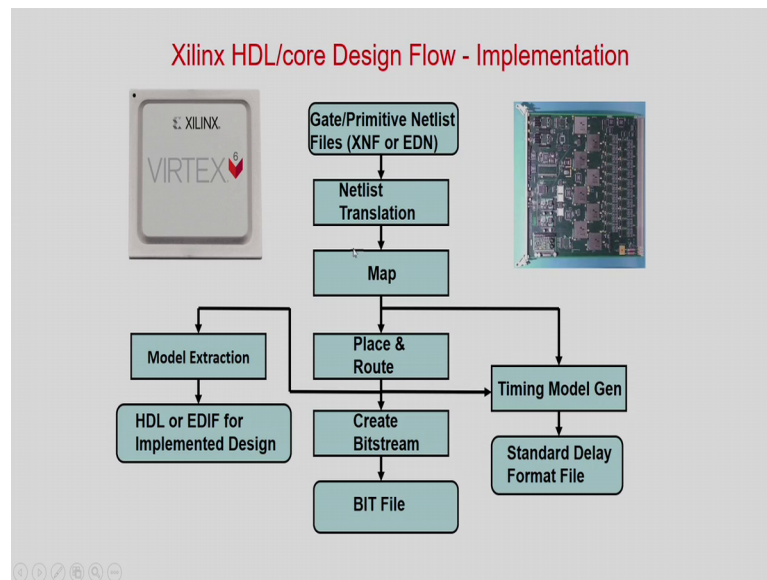


So, I am saying a typical Xilinx HDL flow. So, in this case you write all the HDL files. Then basically some of the synthesis constancy you have to give, because what is the timing requirement, what is this space constrained you are putting and so forth.

Basically, you take takes a target device. What do you mean by target device, because FPGAs lot of variations of FPGAs are there. So, whether you are taking a vertex family, whether you are taking a Spartan family, what type of FPGA configurations you give you have to tell that in this FPGA, I want to synthesize.

Basically all implement because some of the FPGAs are large, somebody have small, somebody are high speed, so you have to tell, which FPGAs I want to use. Then there will be it will be synthesizing your circuits not in terms of gates, but in terms of LUTs on the target device.

(Refer Slide Time: 28:32)



Then basically what you have to do with this target, because now you have all the LUTs targeted. Then basically, there is an automatic map and place route. So, this is actually synthesis flow. In the ASIC synthesis flow, basically you get a gate level implementation, here we will get a CLB level of implementation.

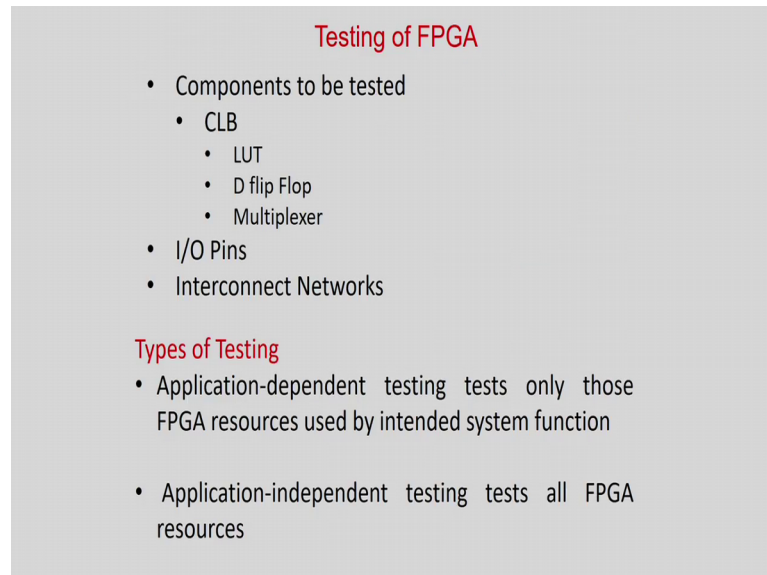
Then basically with the CLBs, you have automatic implementation tools for the for Xilinx from Xilinx and all other vendors. So, basically they will take your basic so the net list basically, which are nothing but LUTs, they will map it to the respective CLBs, they will place and route as I have shown you. And finally, they will generate a bit stream, which will be downloaded, and it will become your hardware.

Of course, I am not going into the designed flow much, because this is not a course on a VLSI design, but this more or less is the flow. So, two figures I have shown you this one example of Xilinx FPGA. And this board as you can see, they are not using for prototyping, rather they have put lots basically I let me zoom it for you.

So, rather they have put lots of FPGAs as you can see from Xilinx, make it can be from any vendor, they are putting it. And it actually sold as a real product that means, these FPGAs have some functionalities as a hardware, which is not reconfigured. If they have downloaded it, they have verified it to be proper, and they are not is not used for a prototyping. Basically, they have (Refer Time: 29:47) feed functionality, and they put it as a as

if is an ASIC permanent hardware and sell it to the market, so that is why, the test of a FPGAs are becoming very very important.

(Refer Slide Time: 29:56)



Testing of FPGA

- Components to be tested
 - CLB
 - LUT
 - D flip Flop
 - Multiplexer
 - I/O Pins
 - Interconnect Networks

Types of Testing

- Application-dependent testing tests only those FPGA resources used by intended system function
- Application-independent testing tests all FPGA resources

Now, let us see let us try to focus, and try to see what is the philosophy difference between the normal circuit test and a FPGA test. First all I could say that as I all a stuck at fault model is a very well fault model. So, why should not I go for it stuck at fault model or bridging fault model answer is not possible, because there are no gates.

So, in fact there all memories and multiplexers, and basically some kind of a LUTs, so that type of stuck at fault model and bridging fault model, whatever I have told you is not at all applicable over here. Secondly, observe ability and controllability are very easy in FPGA why, because this is full reconfigurable. You take one block, make it test the other block making analyzer, and switch it something like bist.

So, in this case it is under test and this is under analysis, then again maybe it can be test. So, both these blocks under test will give the answers to the third block, it will analyze the response, then you change the roles. So, here and here basically as I showed you ATPG is very very trivial, because the gates are very simple. There is a LUT, there is a D flip flop, and there is a multiplexer. If you look at the CLB, it is only three stuff, LUT, D flip flop, and multiplexer, and there are some interconnection.

So, tell and all the boxes are very very symmetric. So, it is very very ATPG is of course is very very trivial by doing some 12 fluids with me as off now. I think you might have one of very simple idea that in this case the ATPG will be very very trivial test pattern generation will be extremely trivial procedure.

And also testing is also very simple, because you can maybe you can put some kind of another block to put the patterns TP generation, you can put over here. The same values can be given over here, because another advantage is boxes symmetric, they are same blocks. I can put some test patterns from this block to the block under test, the same one I can fit another block under test. And the response I can put the analyzer block, which is the simple XOR, and it should be able to find out that both of the blocks under test should give the same output.

Because, basically the blocks are same in function so, you just give same inputs take the outputs XOR it, and you should get same results, and your job is done. So, whole ATPG of or testing of FPGAs are based on the simple philosophy. Gave some give some simpler inputs to the both the blocks under test, and they should give the same output.

Instead of two blocks, you can give 3, 4, 5, 6 10 blocks, and they should give the same answers, which ever block is giving a wrong answer, you can tag it having a error. But, the problem is re-configurability, because every time as I am saying this is a block, which will generate the pattern this is the testing block, testing block, this is the analysis block. So, I have to download that configuration, so that these blocks give reoriented in this fashion.

Now, once this testing has to be done, I have to reconfigure. So, in this case may be this one will be another test block, this may be the analysis block, and may be this may be the pattern generated block, and this may be another block another test. So, role will be changing, orientation will be changing. So, every time one part of one testing is done you have to again erase the FPGA, download a new bit file, so they can reconfiguration is done, and again is test, and you are going to test it.

So, in FPGA the philosophies slightly different, FPGA is simple test application time analysis is simple. Only thing is that if you have to change the configuration of the FPGA for test that going to take long time, because FPGA tests basically takes more amount of time, because is not about the applying of the test pattern or test pattern

generation is because from mode a 2, mode b 2, mode c different base of orientation as I have shown you, you have to do it to test the different different components. And from one configuration to another configuration test means you have to erase the FPGA, download a bit file that takes time.

So, now some people argue that FPGA testing should not be done at the ATE level rather you can purchase it, and you can do the tests, at your at the user location, so there are lot of philosophy. But, if some FPGAs are very high and expensive FPGAs, then you have to of course with that the ate and then basically you have to CP to the customer. Because, FPGAs are mainly for prototyping, you will download the codes, you can test your FPGA, and then you can sell it into the market. So, lot of flexibility here.

So, many people say that I let the process of downloading, reconfiguration testing be done at the use on it, because then you test an FPGA, do a everything then you download your staff everything is operating fine, then you sell it into the market. So, some philosophy someone some people tell it that you have to do in this way.

If the FPGA is chip, but some other people also say that is a very high end FPGA, then it is quite expensive, then it should be done at the ate level before selling into the market. Again there are two types of FPGA base testing, one is called application-dependent, one is called application-independent. What is application-dependent that is I take an FPGA, I download my required circuit, then and then it will be become my hardware, you test that that is called application dependent.

Another is called application-independent that the raw FPGA is there, you will verify that all the blocks are proper or not. So, the more majority of the researcher researchers are working on something called application-independent testing, you get a raw FPGA quickly test it. Because, if you are going to test an application-dependent circuit, then basically it is a circuit, you cannot do much reorientation, because the circuit is there. If you test the circuit, whether is operating properly on the FPGA or not.

So, at some parts of FPGA will be not be tested. And some of the parts of the FPGA you should must not test it, because for some particular application certain configurations may not be valid. Main difficulty here is that the these there cannot be any structural testing in FPGA, you have to note the point that application dependent testing is difficult, because it is totally dependent on the circuit itself.

In case of ASIC, we were doing structural testing. Structure testing means, you take a fault model and you do the test. But, stuck at fault model, bridging fault model are not holding fine in FPGA. So, therefore you have to go for functional testing. So, if this is a functional testing in case of application-dependent FPGA testing, then the designer has to be there. Designer we will look at the design, if you will give a test bench, and that is basically nothing just like your testing your C code, so basically you will be having the FPGA, downloading the software into the FPGA, and just like I am doing a software testing if you do the hardware of the FPGA for the test.

Just like if you see all the loops are come order or not accept some of the philosophy, we will look at tomorrow in the next lecture. When I just like normal software test, you download the hardware in the FPGA, and that code basically you test, just like because there are a very log code or VSDL code, which is download to the FPGA, you test the code. But, not in the soft level, after it is download to the hardware.

So, therefore it is a kind of full software base testing, more or less software base testing, and you require full designer knowledge for that. Because, stuck at fault model or the bridging fault model, which we know basically no longer that way validate much valid over such kind of application dependent test. But, most of the test engineers are interested in application-independent test. What is an application-independent test, raw FPGA testing. So, therefore test engineers are more interested on that. Today, we are going to look at application independent test.

(Refer Slide Time: 36:30)

Defect Locations in FPGA

- *Interconnect defect*: this defect can be usually modeled by bridging faults or/and stuck-at faults.
- *CLB defect*: The CLB is the basic unit of a FPGA chip. It usually contains a built-in Look-Up Table (LUT), which can be configured to realize any possible combinational output for a given number of inputs. A faulty CLB can be detected through the functional test of the CLB. Similar, for the Multiplexers and Flip Flop.
- *IOB defect*: If the IOBs are faulty, the information exchange with other components in the system may not be possible or reliable. Stuck at fault model suffices for I/O blocks. Simpler to test because of the easy controllability / observability

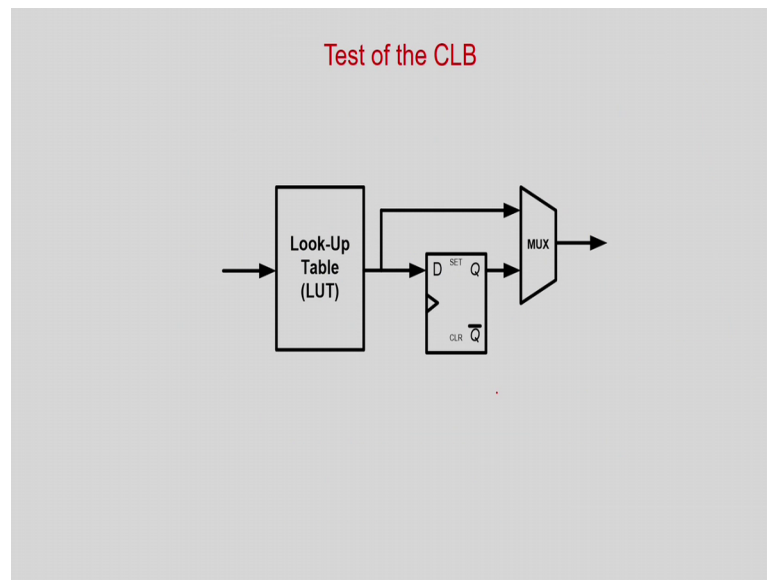
So, therefore what are the different blocks in FPGA. So, there is the interconnection, so there can be interconnect defect. So, interconnect testing is very simple, there are two wires. So, you send a zero and zero should come you send one, one and one should come. So, generally what happens either two lines or three lines get stuck together or the lines get opened, so then they call it open and stuck type of a faults.

So, basically testing off these interconnects simple model, because anyway we are not going into details full FPGA testing, because that can be another course in itself. So, we are just going to take very simple stuck that wires are there you send a 0, you receive a 0, send a 1 receive a 1.

CLB, so you have to test the three components the LUT, the flip flop, and your multiplexer, because you have to test it basically. So, it is very simple in most of the case, again I am repeating the philosophy, to CLBs you take, keep the same inputs, dump the outputs to a loss of basically an XOR gate, and see if they match. So, it is very very simple philosophy of testing, interconnect will be just sending the values here and there.

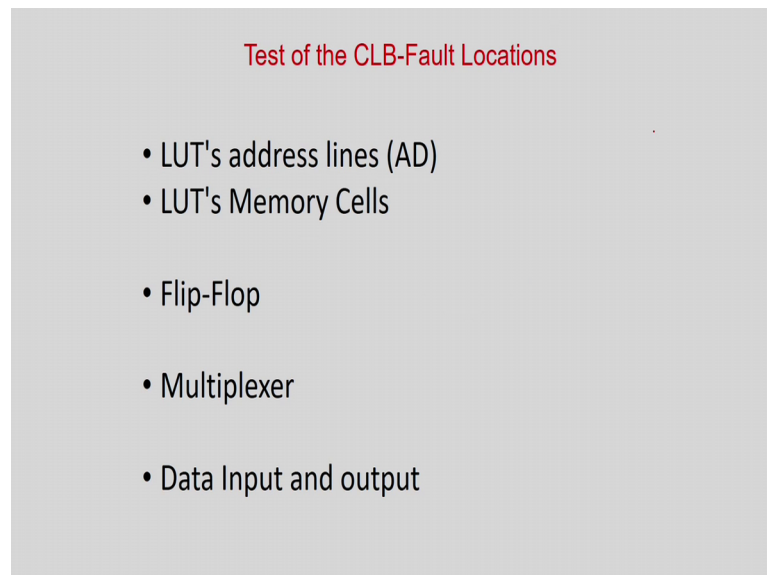
And there is something called IOB defect, basically the if the input, output pins. So, in case of FPGAs the input output pins are also some kind of a blocks. So, there should also be some test principle for that. Anyway, in this course we are mainly limiting up to this, because IO blocks, which scan enables, boundary scans many interesting stuffs are there, but which we are not covering in the course for the sake of brevity.

(Refer Slide Time: 37:51)



So, basically let us come because mainly in FPGA most of the textbooks or lectures you find, mainly we are concentrating on inter connect and CLB level tests. So, this is your CLB. So, what you have to test, you have to test this you have to test this and you have to test this, how to do that.

(Refer Slide Time: 37:56)

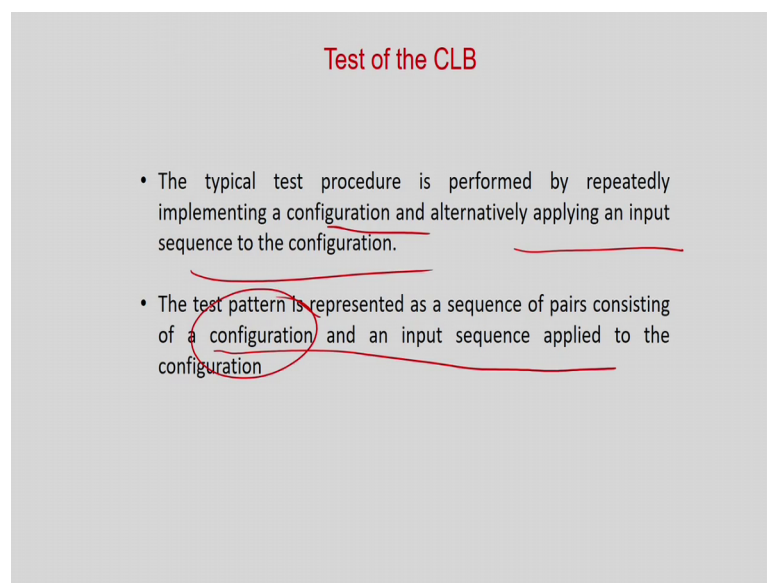


So, one of the different type of the staff in the LUT, you will have this sorry you will have the LUT address lines, and LUT memory cells, it is a memory test a memory. So,

you have to test the address lines that is the address decoder, and basically your memory cells.

So, generally this memory is not very large. In Xilinx may be most of the cases there are 4 4 is to 60 LUTs are available, 4 bit inputs, 1 LUT size is 8. But, in the older generation this one, mainly we had four 2 BIST LUTs, but there are lots of LUTs. So, you combine them to together make a larger function. So, each CLBs are very very simple. Flip flop multiplexer data input output that has to be checked in a CLB.

(Refer Slide Time: 38:34)



Test of the CLB

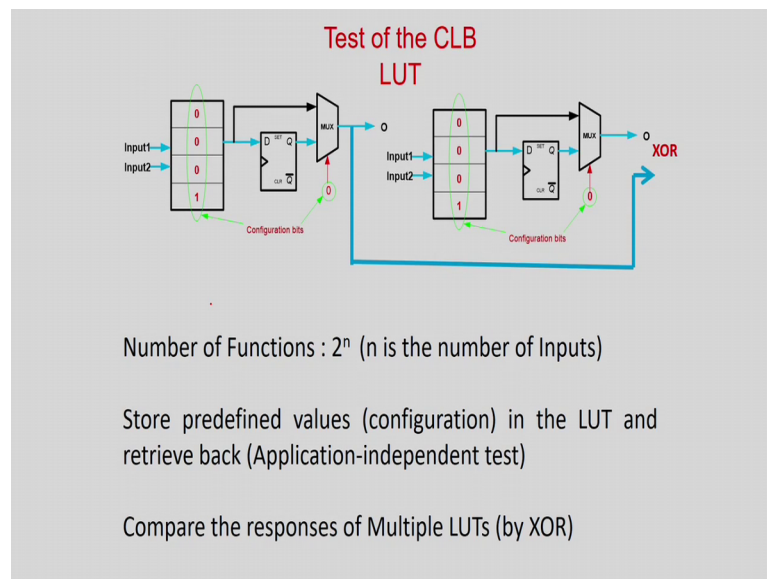
- The typical test procedure is performed by repeatedly implementing a configuration and alternatively applying an input sequence to the configuration.
- The test pattern is represented as a sequence of pairs consisting of a configuration and an input sequence applied to the configuration

So, we typical procedure is very simple; repeatedly by implementing a configuration that is very very important. First you have to configure and alternatively apply an input sequence to be configuration. You take a configuration that this is the two testers. Block under test, you take the outputs put it is an analyze block, some other some other blocks to give the inputs, so there is a configuration. You take the configuration apply the test patterns.

Again basically a test pattern is represents a sequence of pair consisting of a configuration and an input sequence, this is very very I will come back to this. So, basically configuration download a test pattern sorry applied a test pattern see what happens rearrange it. Then again put another test pattern, and check it. More or less the test patterns are similar. Some inputs check or the take the outputs in a XOR, XOR series see, if it is similar.

So, here more important is a configuration. So, in this when you call about test of FPGA, basically it means two things. Input sequence there is the test pattern as well as the configuration, because configuration here actually takes the main time. So, most of the research in FPGA testing is how to make intelligent configuration, so that this testing can be done faster. Anywhere as I told you we cannot tell about the full FPGA testing into lectures, whether in this well try to give you some main ideas or will a preliminary thought, then how what configurations, and how it has been can be done for testing.

(Refer Slide Time: 39:45)



So, it is a simple logic where again taking. So, let us test this two LUT 1 1 0, 1 1 which you have taken. So, if you look at if you look at a very general sense, so if you take a very general sense, they are 2 inputs and basically 4, it is a 2 is to 4 memory. So, how many functions can be there put the power force, 16 functions can be there standard theory.

But, should I should I be able to do all those tests, you can do it, but it will take more time. So, basically you have to decide that same inputs. So, whatever input I will give to this, same input I will be giving to this. I will take the output for this MUX, I will take the output and I will put a XOR that is what is the idea. Same inputs, I will giving I will be giving to the both the CLBs outputs will be there, and I will check it.

So, as I told you for most of the cases the inputs are 2, but for high in FPGAs maximum 4 inputs for a single CLB. So, therefore checking a seeing single CLB is not very much

complex basically, because the number of inputs are not too high. So, maybe 16. So, there can be 2 to the power 16 different functions, but still people you are try to explain all possible functions.

All possible functions means, this is one function. So, this can be all zeros can be one functions. So, n number of different implementations can be possible by a 4 bit 2 to 4 bit memory, 16 different functions can be there. But, people they need do not go about a you have to decide what type of vectors can be there. May be all zeros, I will give over here. And I will check that this bit all the matching's are there. May be I will put 0 1 1 0 as the this one is 1 0 1 0 get also 1 0 1 0.

And I will do the test, test means what? I will first access this bit, I will access this bit sorry access this bit, and there should be equal. Next, I will axis this bit, I will axis this bit, it will be similar, and I will go for the 4 4 4 memory locations. So, it should be similar. But, again you thing is that I cannot if I test for all zeros over here all zeros for a the average may not be high, so I can be testing for all 0s, for all 1s, alternative 0s, alternative 1s that is generally what people do.

But, most of the cases first they will test with all 0s, then they will test with all 1s. Generally, two patterns they take. Pattern sensitivity basically, as I told you can take 0 1 0 1, and same thing you can try over here. So, different combinations you can try. But, generally if you recall more rigorous, then you can try different combinations are the inputs. But, most of the cases if this surprises with 0 0 0 0 1 1 1 1, important is configuration.

So, this is simple. You apply you keep all this series maybe this series, and you start with 0 0 0 0, so here the answer should be this should be a 0 over here, this 0 should come over and XOR should match. So, 4 combination 0 0 0 1 1 0 1 1, but after this configuration has been tested; I have put try with another configuration maybe the other configuration will be 1 1 1 0 this configuration, I may be trying to test.

So, now important is that, it is very easy to apply inputs pattern as 0 0 0 1 1 0 1 1. But, you want to change the content of the CLB, then the whole FPGA has to be erase, and new data has to be downloaded, so that the value will be now 1 1 1 0 1 1 0, and that can be tested. So, therefore in this case applying this patterns is not very important, it is simple very very simple. But, the whatever the value you want put into the CLB for test

that has to be done by a reconfiguration and reconfiguration of very very time consuming process.

But, what is the another advantage? Another advantage is basically FPGA difference CLBs can be test at a parallel. So, maybe this a this is actually n cross n , why it large FPGAs happen. So, maybe these are very large components. So, this one you are maybe in maybe in for test, this was the test, this is for analysis.

Similarly, this one also can be parallely tested, analysis tested right. So, this is another for tested. So, the idea is that the FPGA I can with the CLBs of the FPGA can be divided into chunks. And basically, they can we actually parallely tested or concurrently tested that is why, if testing is passed. If it to have been the case that you have to take to two CLB tested, erase it again download average pattern test it, then it would take they have taken years to test the FPGA.

So, the advantage of FPGA is concurrent test, which is not possible in a what do I see in a (Refer Time: 43:52) in a normal ASIC, whole ASIC has to be tested together in sequence. But, in this case you make a cluster of different CLBs of the FPGA, which have to be tested. And for each of the clusters, you can of test parallel like the example I have given you.

Now, once may be with this sequence you have tested, you have to try for different sequence called different combination in the LUT. Then you erase the whole FPGA, download different configurations. Maybe with the same clustering, but you change the values, so that is why, it is not very difficult. If you think that, you will take a long time to test the FPGA. Because, if you have tested for one sequence, then you have to test for another sequence, everyone this input combination can be given externally, which can be very fast.

But, this is what is the content also should be change, change that content means reprogramming that may make a long amount of time. But, again to the FPGA architecture things can be done concurrently. Once set here, once set here, once set here, you make a cluster of the CLBs, what to be tested, and what to be analyzed make some common clusters, and test in parallel. The idea is stored predefined values in the LUT, and retrieve back. There is actually called application-independent responses. Compare the responses with multiple compare the responses of multiple LUTs by XOR gates.

So, you take you can take 10 testers together and 1 or 2 will CLB, you can make as a analysis block. And you just you make some clusters and you repeat it, so that the testing happens in concurrency. How you can take the D flip flop is also very very simple, only thing is that basically in this case the configurations is to be 1. So, maybe 0 0 0 1, so instead of directly sending as the flip flop you have to apply a clock, So the value will be coming out to do multiplexer. Same thing you have to make is a one, this value will come toward towards the flip flop, and you have to go for an XOR.

So, it is very simple. Just like the normal what do I say, basically you have to you are taking the CLB, just taking the just testing CLB. Here this is just the same thing instead of directly putting the CLB value as a output, you would just basically send it through flip flop. So, just very very simple, only thing is that you require a clock, and there should be done synchronously.

(Refer Slide Time: 45:56)

**Test of the CLB
Multiplexer**

Normal behavior	Faulty behavior 1	Faulty behavior 2	Faulty behavior 3
begin	begin	begin	begin
if(condition)	if(condition)	if(condition)	if(condition)
Output=input1;	Output=input2;	Output=input1;	Output=input2;
else	else	else	else
Output=input2;	Output=input1;	Output=input1;	Output=input2;
end	end	end	end

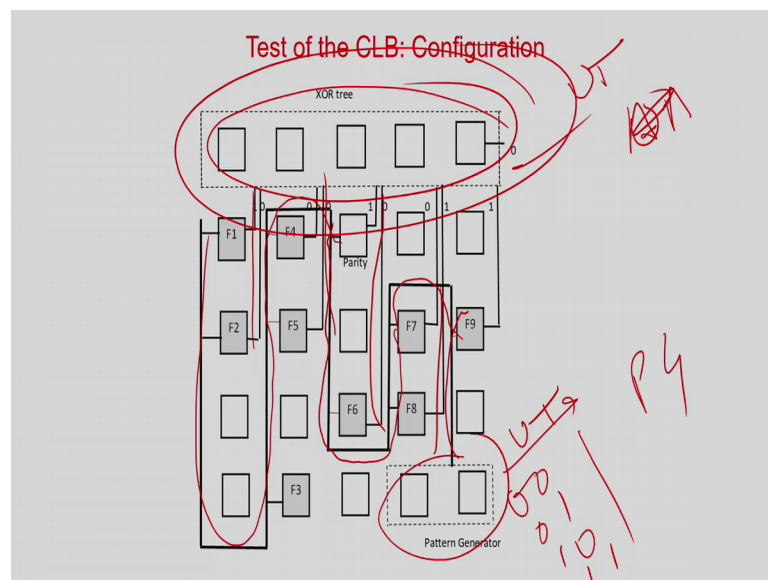
So, testing of the flip flop as well as your CLB is very simply done. What about the multiplexor is trivial as you know that this is what is a multiplexer fault model. Already we have discussed at length, when you were discussing about high level fault models. And basically, it says that the conditions may be inter switched single condition may come out the second condition main causes output as the permanent values.

But, in this case, we have will check for all combinations like 0 0 0 0 1 1 0 1 1 1, so maybe basically someone fault model is that in all cases this what is going to come out,

this is never going to come out. So, but anyway if I am what about the test I have told you, for MUX sense CLB if you do it, so the LUT you do it. Of course, both the lines of the input lines of MUX will exercise, also the configuration bit has been exercise though MUX is auto tested.

So, if you test for the flip flop, and if you test the CLB sorry the LUT automatically the MUX is already tested, because this is the well known fault model of the multiplexer. So, as of now what we have see the idea is very simple that basically test patterns are very simple. Only it is non-trivial that how you can apply to all the patterns or this is more challenging that how you may clusters.

(Refer Slide Time: 46:56)



Like for example, this one will actually clear it. So, in this case maybe, he is trying to test concurrently number 7 8 all the flip all the CLBs if you can see, which are in grey color are been testing in concurrency. So, this is a pattern generator.

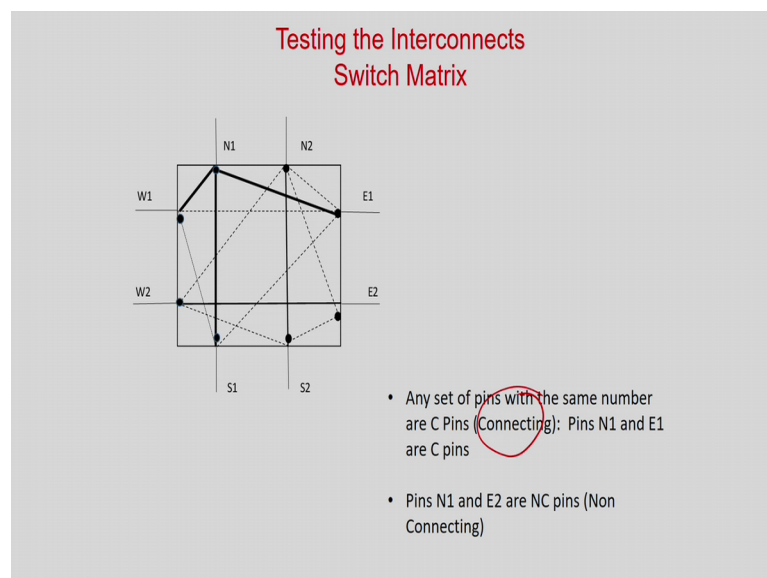
So, pattern generator in this case is nothing but 0 0 0 1 1 0 1 1, all the BIST of the CLB, it is excessing. So, this is going to has an input to 7 the same is going to input as 9, 7, 8 the thick line basically it is going to the input of 4, then it is coming back an input to 1. So, is same input have been given to 7, 9, 7, 8, 6, 4, 5, 1, 2 and 3. So, all this gray color FPGAs CLBs same input has been given by the pattern generator is nothing but 0 0 0 1 1 1 0 and 1 1 this is something like, these are the cases; so, one generating this all possible combinations.

So, all the values an output of all these CLB are actually fed to XOR tree. Some blocks have been configure to an XOR block. Every if everything is normal, everybody good should give the same output, which has been told as the LUT. So, if everything may check, all the XORs are 0 fine. Any error somebody some of the some of the values are different there is a problem with the CLB.

So, maybe I can it is a one way of testing. Some of the CLBs, I am converting in XOR gate, which is are not tested. These are the analyzers. And the pattern generator some of the CLBs, I am using to the generated pattern generator, so it is call the PG. So, this is one way of clustering. Now, if we again change rule, so this one will be now under test, maybe this also this will be under test, this one will also under test, second this one will be under test, this one will be under test, all the blocks which have been tested now should be converted into analyzer and pattern generator.

So, basically that is actual call configuration changing. Again repeating in case of FPGA testing pattern generation and application is very simple. Same patterns we applied to multiple block, you should get this same answer. And there is can be XOR is to do it. But, again you have reorient among themselves you change 0. So, you have to do reconfiguration. Reconfiguration means erase the FPGA, download the bit file, which takes actually takes time. So, here the philosophy is slightly different.

(Refer Slide Time: 48:53)



Now, last but not the least if you check this switching matrix so, switching matrix is very interesting. So, if you look at the this the switching matrix, so in this case as I told you the lines can go over here, it can go over here or it can come over here it can come over here, it is a 2 cross 2 switches, n cross n switches are also there, 4 cross 4 switches are also there. Similarly, this line can go over by this way, similarly this can be going in by this way, similarly also we can have something like this one can be going by these lines.

So, basically this dotted lines are showing, which pins can be connected to with pins. So, as I it is something like the I means any line can be going by this, it can be going to this, it can be also going to this. So, any line can be going in these four directions. So, it is a simple nomenclature they use west, north, east, south. And if they say that E 1 can be connected to only other one, so it can be connected to this, it can be connected to this, and it can be connected to this.

So, anyone it can be connected. E 1 cannot be connected to S 2, it cannot be connected to W 2, so that is a simple way of handling. By and similarly for all like N 1 can be this bold lines you can see N 1 can be connected to here, connected to here, connected to here. So, N can be going in the three directions possible. So, for all the let us the same thing is possible, so it is a 2 by 2 switch, 4 by 4 switches are also available, 1 by 1 switches are also available right.

So, in this case it can be connected to any of the four direction. So, the all the numbers with same numbers are all connecting pins, and where it cannot go like E 1 cannot be connected to is to they are call non-connecting pins. Now, I have to just do see whether these connectivity matrix as well as the interconnection fine all. It is very simple. You make a connection all possible connections you make, and then you apply 0s and 1s you get it.

(Refer Slide Time: 50:36)

Testing the Interconnects Fault Models

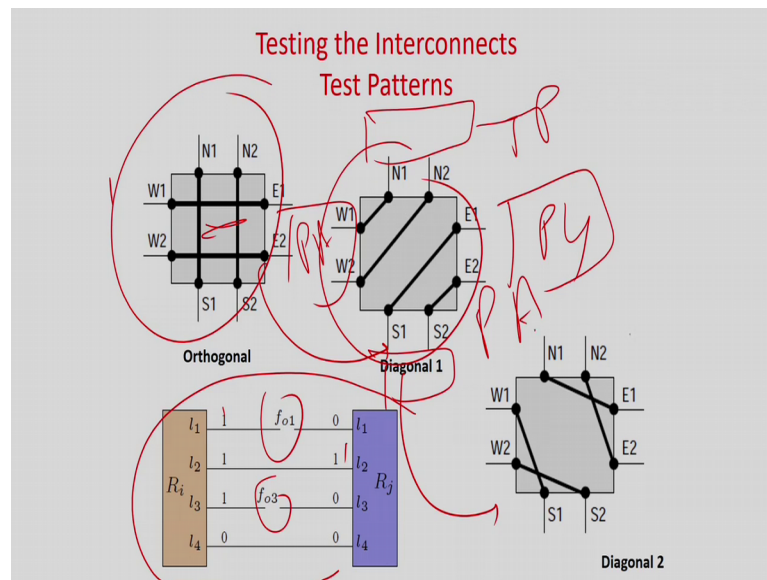
- *Permanent connection (PC)*. A short on any pair of NC pins.
- For the C pins, we need two fault models, depending on the FPGA configuration:
 - *Permanent connection (PC)*. A bridge of any pair of C pins.
 - *Permanent disconnection (PD)*. An open on any pair of C pins

Now, you may think there is a quite complicated type of connections, lot of connections can be possible, lot of permutations can be possible, interestingly low. Basically, (Refer Time: 50:38) before going to that let me tell of the fault models, one is called the permanent connection, and one is called the permanent disconnection.

So, what is mean by permanent connection that means, these two will be permanently short, you cannot remove the connection. Permanent this connection means, you want to make this connection, but it will not be connected. So, this may be between the comfortable lines. So, to these are two comfortable line N 1 and S 1 may be directly connected or you may not be able to connect any point of time.

Also there can be a fault, this is a permanent fault kind of a thing. If say this one gets connected to this, although I do not allow N 1 connect to S 2, it can be also connected to this point that is that can also be an error. So, this is called permanent connection a short between any to non-connecting pairs, and with into comfortable lines there can be open. So, you want connecting not connecting and permanent connection means, you want to open it, it is not opening.

(Refer Slide Time: 51:34)



Now, very interestingly all combinations must not be tried or whether need not be tried, only by three configurations everything is possible. What are the three configurations, it is called orthogonal you are by straight. One is by diagonal, and this is call actually diagonal two diff of connection. Why three, basically if you look at, this point can be connected to here, this point can be connected to here or this point can be connected to here. There is a only three ways are connection can be possible, similar for the all other nodes.

So, I should able to see if all the possible connections are exercise properly or not that means, I will give 1, I should get 1, I should give a 0, I should get 0 that is it. Similarly, 0 0 0 1 1 1, I should be able repeat it for all. So, sequential again the beauty of FPGA testing is that, sequentially it is not required. Parallely, I can check 0 0, here also 0 0 0 0 I should be able to get it. Similarly, I will be parallely, I will put 1 1, I should get a 1, put 1, I get a 1 I put a 1 put a 1, I get a 1, I get a 1 similarly sorry I can get it parallely.

So, in one configuration all test, and it is will be take checked. Next, I will go for this connection and apply parallel testing, putting all the 1 1 1 1 and get the value put 0, and this is another connection. So, by three connections, three different configuration, I will be able to test the all types of faults permanent disconnection, permanent connection. Permanent disconnection means, what you apply a 1 over here basically, your

disconnected you are going to get a 0 over here, and that is it actually called a disconnection base test right.

So, I am not going in to the very detailed theory, why only these three is there, there mean formal proves that tells that by only these three connections, you can test all the different type of fault models in case of these interconnections. But, I think I have been able to give you a philosophy that a point can be connected to either this or this or this.

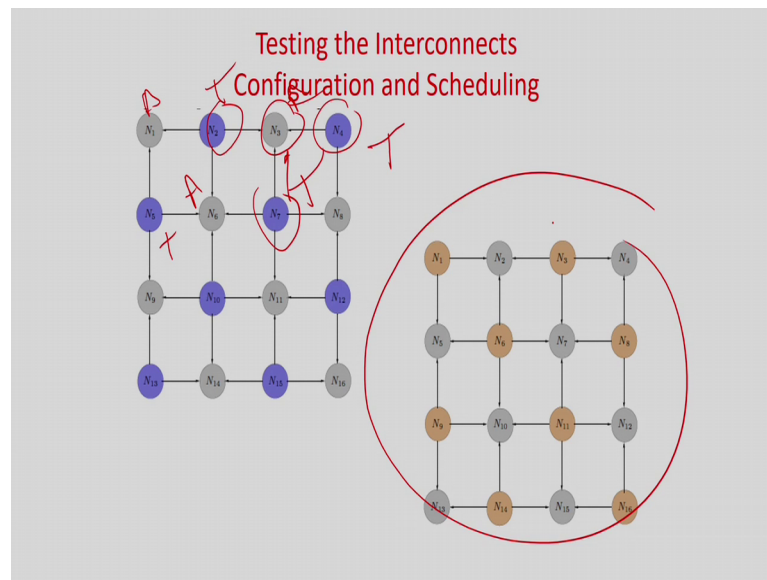
And I by applying 0s and 1s, it properly they are being transmitted to the counterpart all faults are basically kind tested. And I can do it in concurrence this one, this be configuration, this configuration, and this configuration are three different configurations. But, I can take this line take this line, this line, this line in one go. Similarly, this I can testing one go, and these four I can test in one go that is the beauty of FPGA testing, you can apply concurrency.

Now, this has been shown basically in this slide that what is simple that is a sender and a receiver. You send a 1, you should receive 1. You send a 0, you should been receive a 0, then the call open and all you are not going to get it that simple analysis can be done. So, this may be one CLB, this may be another CLB, this is pattern generator, and this is pattern analyzer. So, similarly this can also be PG, and this can be PA, so just you send 1, 0s and your job is basically done.

Of course, there are bringing fault models also there, than if these two lines are bridging and all. So, advance fault models also do exist for this kind of interconnect test. So, slightly complex test patterns will be apply, but again the interesting fact will be very much much simpler compared to normal combinational or sequential ASIC.

Only problem is that whenever you have to go change to go from one configuration to another to another configuration, it will take time because FPGA has to be erase, your bit map has to be download it. But, test application is very simple, because all these LUTs whatever these blocks all the blocks, which have been placed in different parts of your FPGA can be that is connectivity blocks can be tested in parallel that is what is the beauty. Disadvantage is configuration changes are required, it takes time, but beauty is that lot of concurrency can be possible.

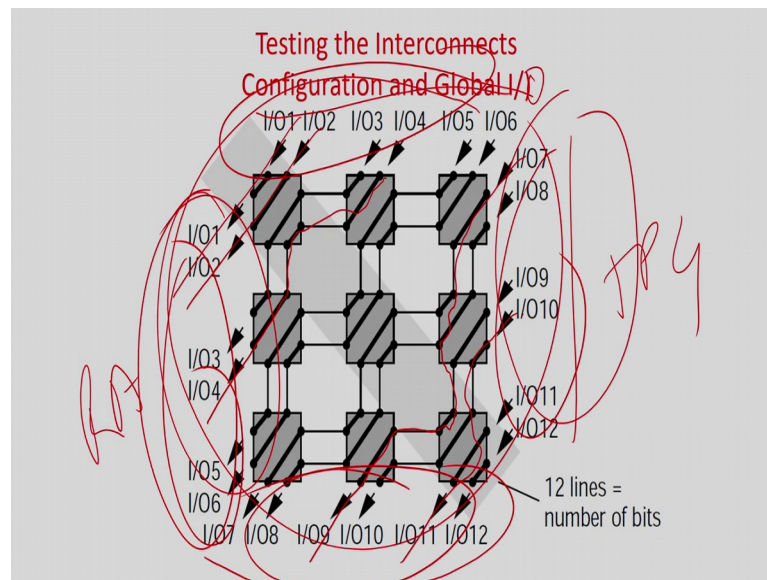
(Refer Slide Time: 54:51)



Like here, I have shown you this one example, this your and somebody's people also call scheduling this is test application as well as scheduling configuration and scheduling basically. For example, say you can see that this guy's giving as the input, this guy's giving the output, and this guy's going to test it. Similarly, this is also giving an input to this guys is blue are on the test, and this guy's actually doing the analysis, this is under test, under test, under analysis.

So, basically which is being test it is testing. So, basically it is analysis, it is analysis. So, this one configuration; in the another configuration, the just the rolls are reverse. So, you have to properly configure and properly schedule at which node will be in test mode, and which node will be in the analysis mode. And you can see because of the symmetry similar test patterns can be given and lot of parallelisms can be obtained.

(Refer Slide Time: 55:35)



So, there is and this one way of testing. Some people also thought in the other way, which is called global input output. Like instead of basically putting these test analysis or everything in a single on the FPGA itself, they try to make a global connection. Like this is one configuration of different connecting blocks. So, you can see this input will be going over here, this will be going over here, this input will be traveling over here, travelling over and coming at the out output. So, this is one way of travelling.

So, in this case basically what they are trying to do, they are trying to basically in this case also you can see, this is going to the input output. Then in what is this matrix basically, they are trying not trying to put any analyzer on the chip itself. What they are trying to, they are trying to make full utility of these blocks, apply the test patterns over here. And get the test pattern out going to the output of the chip while define configurations of your connectivity blocks. So, in this case what is the thought behind this is analysis and test pattern generation, I will keep of chip.

So, in this case what happens basically the idea is that number of configurations will be less. So, here what happens as if is my look at it, so the roles are being continuously change analysis testing, analysis testing, in this case all are under test. So, basically what happens all the test patterns are applied over here. And these are given at the output. So, your ATE is the test pattern generator, this is your TPG and this is your response

analyzer, which is off chip. So, in this case the number of test configuration will be less, but again are nothing is a magic in any science.

So, in this case what happens basically test pattern generation is complex, there can be lot of controllability, observability problem, just like because is a now become a combinational or a sequential circuit. Because, test patterns are applied from the output input and the out and the values go out the output any and you have to go a match in the ATE. So, normal become a normal sequential combinational circuit test. Problem here is that there is no standard fault model, like your stuck at fault model or bridging fault model etcetera.

So, even if you say that configuration global IO, you can use a technique when you can you should not chase the rows like in the previous approach basically some block is tested, some block is analyzer; so, in this case to avoid such reconfiguration. All the inputs are given from the ATE proper routing is there, and all the answers are thrown out of the chip, so that you analyze outside.

But, again as I told you here ATPG is more complex, test pattern generation is more complex over here, application is simpler, then analysis also simpler, because they are of chip. But, test pattern generation and basically what values to give, what controllability and observe becomes a major problem. Because, there is no standard stuck at stuck fault model is the more challenging way of handling it, but still I just mention it to you, because this was another direction of thought that if I try to bring every output to the output world, and try to do the test.

The major problem as I told you again one thing, I forgot here basically in that case, here this type of circuit is more suitable. This type of architecture is more suitable, if you are testing for application dependent FPGA. In this case, your one of the beautiful thing we found out of the FPGA testing was a symmetry. Same pattern can be applied to all, so that was actually bringing of the beauty of concurrency base tested.

But, if it is a application-dependent circuit such concurrency will not be there, because all the modules will be implementing different different functions. So, in this case may be this global sending the value, taking out the value will be a easier way of solving the problem. But, again repeating this is not as simple as you are doing in normal sequential and combinational ASICs, because there is standard fault model.

(Refer Slide Time: 58:59)

FPGA based Test: Configuration vs. ATPG

- ATPG is trivial
- Same for all LUTs
 - Parallel testing
- Important: Intelligent Configuration for faster test
- BIST: On-chip TPGs and Response analyzers

So, with this we come to the end of this lecture. So, what you have found, ATPG is very trivial in case of FPGAs, same for all LUTs or all kind of these routers. So, it is simple parallel testing. Important is intelligent configuration for faster test. So, you have to reorient very quickly, and you should to minimal reorientation. So, are in one orientation or in one configuration, you can do maximum testing, so that is why many people and of course, it is more or less BIST kind of an architecture, because on chip test pattern generation resources on chip. Some of these CLBs are configure, some of the CLBs are actually configured into test response analyzer.

So, basically FPGA testing is some kind of a BIST in itself. But, still it can again I should mention in the end that this type of architecture in which case the input or output are from the external world in a FPGA are basically mainly applied for application specific or application dependent FPGA.

So, with this we come to the end of this lecture. And I should again tell you that we are here in this two lectures would be show you the tip of the ice berg, because if embedded system based, programmable devices based FPGA implementation is slightly an newer concept, and lot of works has to be still done or is being undergoing. So, we are in these two lectures, we will not going to the entire picture we cannot give you. But, rather we will try to show you in if you are having reprogrammable or hybrid kind of software

hardware co-implementation in FPGA, in which direction on or what is the paradigm shift require.

So, with this we come to the end again. And the next lecture, we will try to see something more on testing of application dependent FPGAs or if you have a some kind of a processor in the FPGA or in the hardware itself, how it the software implementation can be used to test itself.

So, thank you.