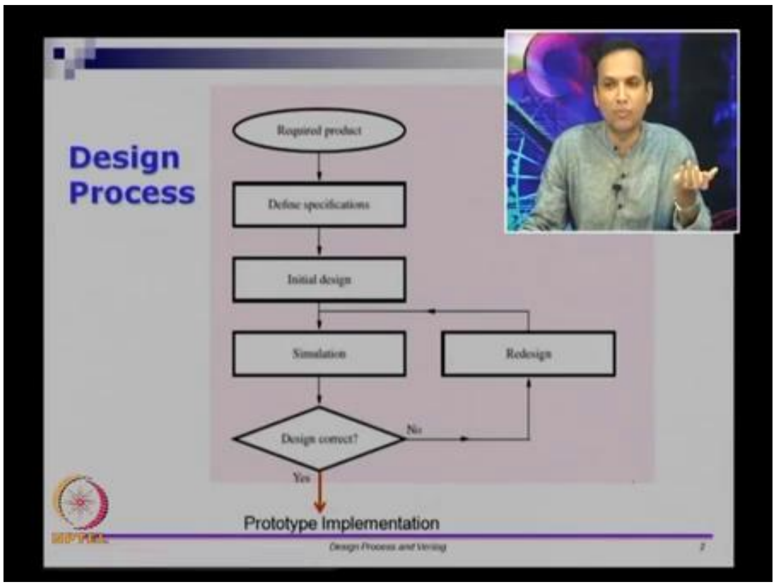
Digital Circuits and Systems

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

Module - 16
Design Process and Verilog

Welcome to module 16. So, in this module, we are going to look at what the design process for digital circuits are and I will also introduce you to verilog. So, let us see the basics of a design process.

(Refer Slide Time: 00:30)



Design Process

So, let us say I want to build a product, so I want to build some product, I start with what are called specifications, I have to say, what this product is suppose to do. So, that forms specifications, this is also something that you do in software. When you design a piece of software you say, this is what the software will do, these are not the thing that it will do and so on.

The same thing is required for doing hardware design, you have to say that this circuit is suppose to implement this functionality, it suppose to run this fast, these are conditions under which you should operate, this is the temperature at which you should be able to run, all of these things you decide. So, for example, if you design a cell phone chip, if you design a processor for a cell phone, then you have to say that it has to be let us say minus 15 degree centigrade to 45 degree centigrade, it should work properly, that is the specification, it has to run at one Giga hertz that is a specification.

So, there are functions that you specify, so there you have to specify functionality, what it is suppose to do, you also give performance constraints, you give things like power constraints, it should run within 1 watt. So, things like that are all constraints. So, you start with all this in mind and you come up with a set of specifications. Once, you have a specifications in place, you go and do an initial design, this initial design we have been doing this using Boolean equation and so on directly.

So, you do the initial design and then once you have an initial design, you go and simulate this. So, when I says simulation, what I mean is, you need to be able to give inputs to it and find out, what the outputs will be without actually realizing the circuits. So, that is what simulation means, so when you do simulation, you have the design with you, you do not have the final product with you. But, when you have the design with you, how do you give inputs to it and how do you find out, whether it is correct or not we will do this in a little while.

So, we do what is called simulation, when we do simulation, if the design is correct, we can then go and do a prototype implementation. But, if the design is not correct, because it is not meeting sum of this specifications, we go back and redesign the circuit or whatever chip that we want to do. So, this simulation is a very interesting thing. So, what we are seeing here is what is called the front end design.

So, you start with something in mind as a specification, you do an initial design of it, if I do it on paper, so let us say I draw diagrams on paper and so on. I can only show it to people around me or maybe I can scan it and send it to you, you can verify it and so on. This is not going to be useful for a computer to be able to take as an input and we cannot give inputs to it and get the power of the computer to give various input and see, what the output will be and so on.

When we design a circuit using computer, what we want is, not just a mechanism by which we can specify what we want, we also want to be able to give inputs to it and see, what output it will be. If I do that in a piece of paper, it is not useful, because a computer tool cannot take this input and process anything with it. So, this, what we are going to do in this course is, we are going to learn, how to do initial design using language called verilog.

So, already mention this in the week 1, verilog is a hardware description language, we are going to do our design in verilog. And this verilog design will be actually at a text based description that a computer can process with appropriate tools. You can actually go and take this verilog design written as a piece of text, you can give inputs to it, you can find out what the outputs will be and so on, all of this you can do.

And you can keep doing this, till you comfortable with the fact that the design is correct. So, when you do verilog base design, at the end of what we see in this chart, you will only get a textual description of the design that you want. Once, you know that the design is correct on paper, you can do that on paper, but here we are going to use tools to ensure that the design is correct. We write them verilog use tools to verify, whether the description is correct or not.

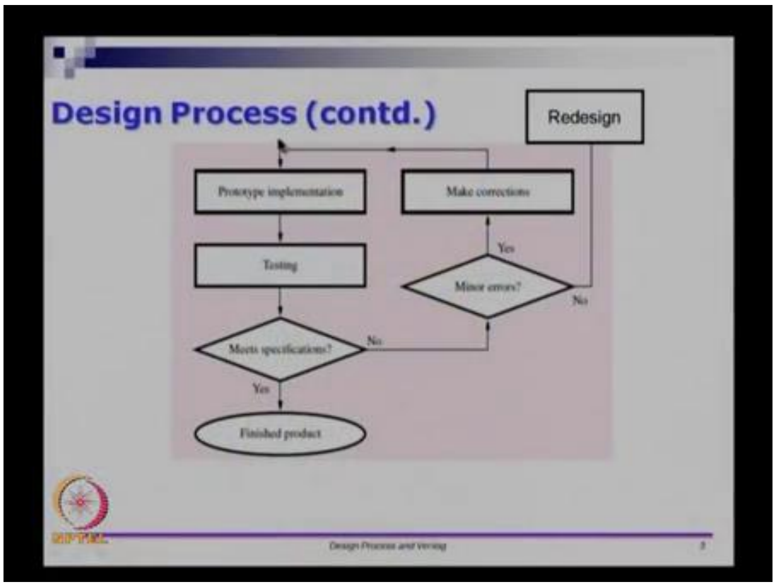
Once, you done with that, then you go and do your prototype implementation. So, till then you do not want to spend money in building prototype, because prototype means money. You have to buy the components, assemble them, spend time, may be put them on the breadboard or put them on the PCP, wire them up, soldered them, whatever, it takes time. You do not want to do this even before you have ensure that you design is correct in.

So, whatever you have in mind, you have to translate that to a process, where you can give inputs and check, whether it is correct or not. So, this is called front end design and this process of doing simulation taking a initial design, writing the design in verilog, do simulation, finding out whether it is correct or not and redesign and loop through it, if the design is not correct. So, this process is called front end design and verification.

You do a front end design using verilog or VHDL, VHDL is another popular language and this process is called verification and verification takes quite a bit of time. So, about 60 to 70 percent of products time is spent in design and verification. So, it is a very important task lot of engineers get employed to do verification. So, if you go and look at companies in India and worldwide, you will see that most of the design houses have a much larger verification team than a design team.

Design teams are smaller, but verification teams are bigger. So, there are plenty of job opportunities, if you know how to do verification well, design and verification well. So, let us continue the process.

(Refer Slide Time: 06:34)



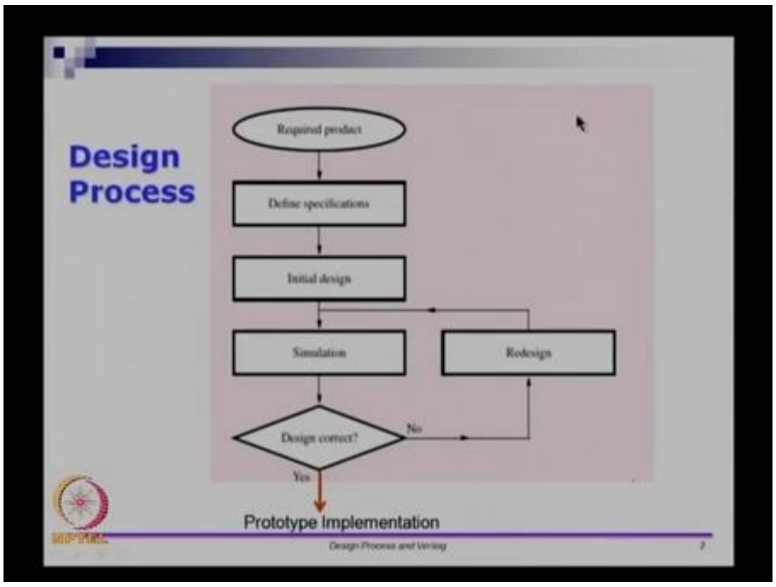
Design Process (contd.)

So, at this point in the beginning here, we have a design which we have simulated in verified, whether it is correct are not. Now, I am ready to do a prototype implementation. So, when you do a prototype implementation, you go and test it, if it meets the specifications, then you have a finish product, so you can now sell it to the market. However, it does not meets specifications, you go and see if there is minor errors.

If there are minor errors, then you go and make corrections to it and change only the prototype, may be there was a manufacturing defect, the design was correct, but the manufacturing was a problem, you are suppose to give a voltage supply of, let us say 1.6 volts, you ended up giving 1.2 volts supply, you have to fix it. So, these problems could come anywhere in manufacturing or in the input that you have given. If there is a minor error which can be fixed in the prototype itself, you do that till you are ready and till, you can guarantee that to product is actually meaning this specification.

But, if we find that it is not a minor error, it is a major error, may be conceptually you have something wrong, the prototype is actually a faithful replication of your design, but the design itself was wrong, if that happen you go back and redesign. So, this redesign is something that may have to do.

(Refer Slide Time: 07:51)



In which case, if you redesigning, you come back here and you again do a simulation, find out, whether this is correct or not, you keep doing this loop, till we again ready for a prototype implementation. So, this process is a very laborious process. So, you do not want to really do this cycle too many times, you do not want to go and redesign, because your prototype had some manufacturing problems are one thing.

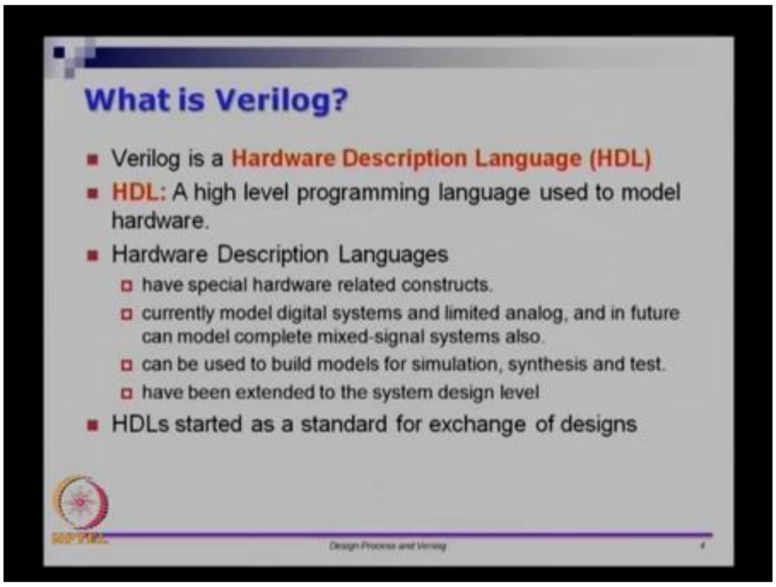
But, if your design is incorrect and if you take it through prototyping, it is a very expensive deal. What you want to do is, stay within this as much as possible, check if everything is correct and once you are sure that this is correct, only then go and do prototype, because prototypes are very expensive. So, I am going to show you, how to do this process.

So, other nice thing about chip design is that this is the one time affair. Coming up with the design and ensuring that verifying that this design is correct is the one time affair. You do prototype implementation, you may actually do multiple prototypes and you go and manufacture it, you get it back and you test it, the testing has to be done for each and every chip that you have. But, the design and verification is actually done just once in the beginning.

Once, you have, it is not that for if I make 1 million chips, I have to verify chips 1 million time. The design is verified once, it is manufacture, when you manufacture you may get1 million chips, these one million chips have to be tested. So, testing is different from verification, verification is something that you do on with software, without necessarily having any chips and so on, you do verification with verilog code or VHDL.

Nowadays, we also have system C, system verilog and so on. But, once you have it, you go and do a prototype implementation, you get it manufactured. But, once you manufacture it, you have multiple chips, you have to test each one of them to ensure that this can be marketed, I do not want chips which do not work to go to the user. So, we would like to get each and every chip tested. So, the first part is called front end design and verification. The second part is called implementation and testing.

(Refer Slide Time: 10:00)



What is Verilog?

- Verilog is a Hardware Description Language (HDL)
- HDL: A high level programming language used to model hardware.
- Hardware Description Languages
  - have special hardware related constructs.
  - currently model digital systems and limited analog, and in future can model complete mixed-signal systems also.
  - can be used to build models for simulation, synthesis and test.
  - have been extended to the system design level
- HDLs started as a standard for exchange of designs

Design Process and Verilog

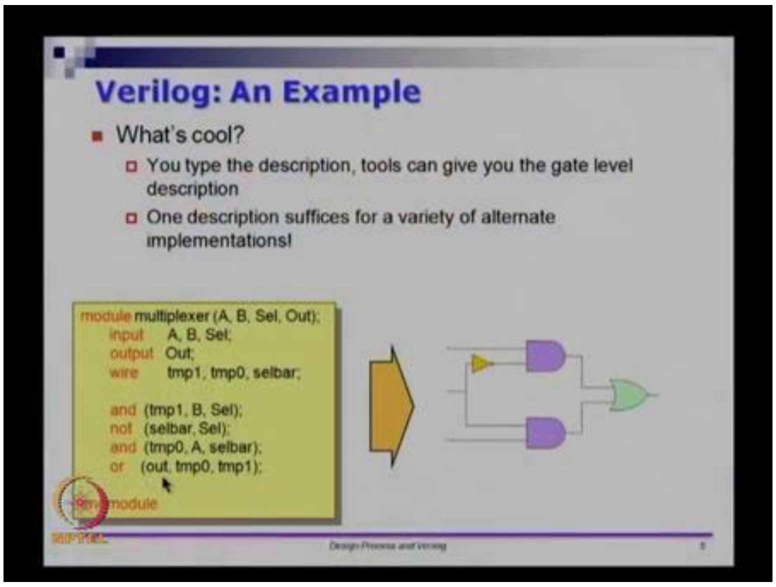
So, what is verilog, verilog is a hardware description language or also called HDL, it is a high level language use to model hardware, I have already flash this slide in week 1. So, they have specific hardware related constructs, they can model digital systems and increasingly analog systems and so on also. And they have been extended to system design level.

So, HDLs, when they started out, they were actually a mechanism by which engineers exchange the designs. So, let us say, I as an engineer brought truth table to do something and you as an engineer brought a circuit diagram to do the same, do some other functionality. And we have to put this together and finally, make a working circuit, I have a truth table, I have to do something and you have circuit drawn a paper and pencil. We have to now go throw a whole process in which we have to interface and get everything working to design a complete system.

So, HDLs actually started as a standard for exchange of design ideals, if I write my description in verilog, you write you description in verilog, we can take these two descriptions put them together and make a larger chip. We do not have to translate from one to the other; that is the main intention with which hardware description languages were invented and it became quite useful after that.

So, instead of just exchange, people started designing new things using HDLs and that is what we will do in this course. Let us take a small example, I already show this example earlier.

(Refer Slide Time: 11:30)



Verilog: An Example

- What's cool?
  - You type the description, tools can give you the gate level description
  - One description suffices for a variety of alternate implementations!

```
module multiplexer (A, B, Sel, Out);
    input   A, B, Sel;
    output  Out;
    wire    tmp1, tmp0, selbar;

    and (tmp1, B, Sel);
    not (selbar, Sel);
    and (tmp0, A, selbar);
    or  (out, tmp0, tmp1);
endmodule
```

Design Process and Verilog

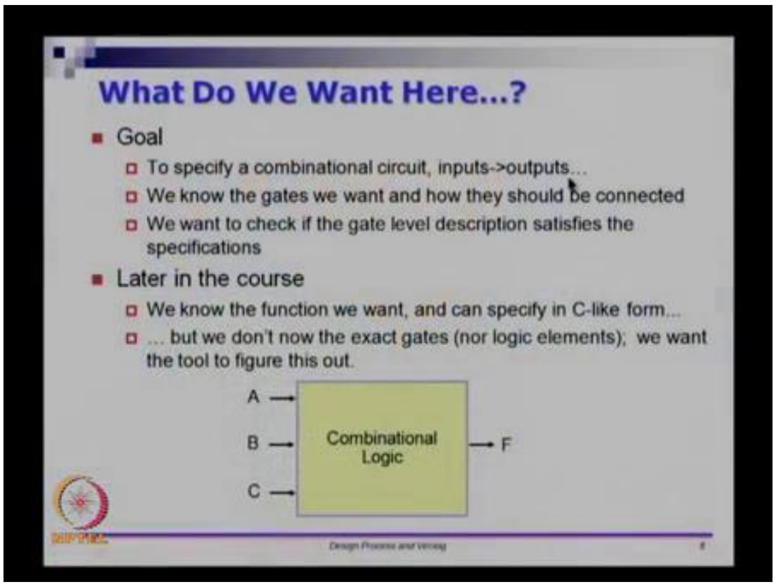
So, what is cool about verilog or VHDL or any such HDL is that, you type the description and there are tools, which can give you the implementation. You write a description like this and there are tools which will give you, what should be the gate equivalents and how they should be connected and so on. And usually, it give one description, you can get a variety of implementations.

So, this arrow mark, there are tools called synthesis tools, which will take your verilog code and generate circuits out of that. The synthesis tools can take the description, they can also take constraints like, what kind of gates you are allowed to use and so on, for example, let say I gave this description and I allow this synthesis tool to use AND and OR and NOT gate, it may end up with something like this.

But, if I tell this synthesis tool that you take the description here; however, you have to use only NAND gates. Then, it will give me a circuit in which all these things will essentially be converted to NAND gates. There are tools which can do that calls synthesis tools, the various companies which do this, which develop these tools. So, that is a very cool thing to have, you can change the implementation without changing the design that you have.

And the other major thing with the verilog is even though this design is actually written in terms of AND gate, NOT gate and so on and describe this in week 1, you do not have to describe at this level, you can make it even more abstract. We will see things as we go alone, but for this video, you are going to see only something which is describes in terms of basic gates.

(Refer Slide Time: 13:05)



What Do We Want Here...?

- Goal
  - To specify a combinational circuit, inputs->outputs...
  - We know the gates we want and how they should be connected
  - We want to check if the gate level description satisfies the specifications
- Later in the course
  - We know the function we want, and can specify in C-like form...
  - ... but we don't now the exact gates (nor logic elements); we want the tool to figure this out.

So, what do we want here we want to specify a combinational circuits, which means given the inputs, we should be able to derive, what the outputs are. And at this stage, let assume that, we know the gates we want. We have done something in a piece of paper, we have evaluated everything. We have minimize may be and we have Boolean equations or even the circuit ready and I want to implement, I want to represent this in verilog.

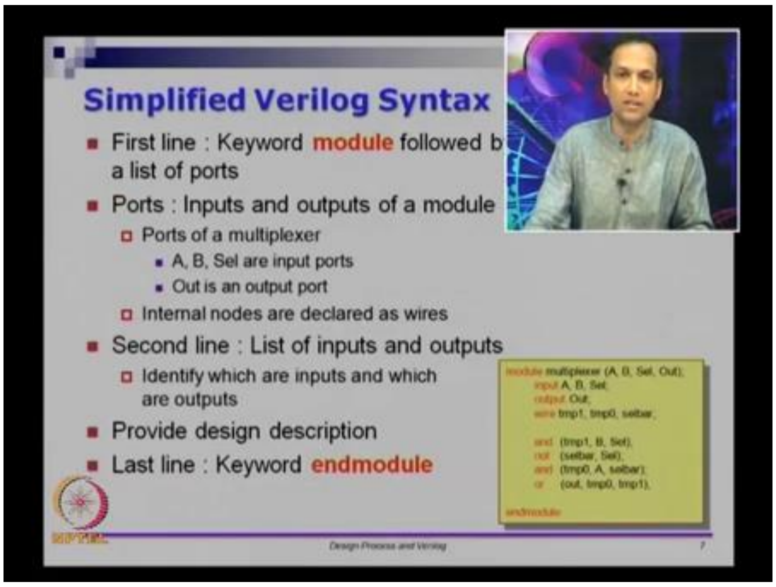
So, this is not the ideal way in which verilog should be used, but this just a logical step from wherever we are right now. So, we know how to take a description, we know, how to write truth tables, we know, how to get basic gates for it. Given the basic gates, how do we write verilog which will represent the basic gates on the connection. So, what we have going to do is, we know the gate that we want, you want to check, if the gate level description satisfies the specifications are not.

You came up with the circuit, you can write in verilog and check, whether the circuits correct or not. Till now, you are actually plugging in various 0's and 1's and checking; now you can actually use the power of software and tools and computers to do this. Later in the course what we do is, we may not even what gates we suppose to use, we know the functionality.

We know, what the function is suppose to be and we will specify it in like language like C or C plus, plus are fairly abstract. We do not know, what they extract gates are, the logical elements are and we want to let the tools to figure it out, you can do that also in verilog. So, right now what we are going do is, we are going to assume that, I want the implementation function F in terms of let say 3 inputs A, B and C.

We will need a mechanism by which want to represent the logic which will get F in this video, this combinational logic is not something which is very abstract we will describe it in terms of gates. But, later, we will see, how do write it in an abstract way.

(Refer Slide Time: 14:55)



Simplified Verilog Syntax

- First line : Keyword **module** followed b a list of ports
- Ports : Inputs and outputs of a module
  - Ports of a multiplexer
    - A, B, Sel are input ports
    - Out is an output port
  - Internal nodes are declared as wires
- Second line : List of inputs and outputs
  - Identify which are inputs and which are outputs
- Provide design description
- Last line : Keyword **endmodule**

```
module multiplexer (A, B, Sel, Out);
       input A, B, Sel;
       output Out;
       wire tmp1, tmp0, selbar;

       and  (tmp1, B, Sel);
       not  (selbar, Sel);
       and  (tmp0, A, selbar);
       or   (out, tmp0, tmp1);

endmodule
```

Design Process and Verilog 7

So, let see verilog syntax, so you see this in the bottom here that the description of multiplexer done using verilog. So, the first line starts with module and the last line ends with end module. So, whatever you see in red here are actually keywords, keywords meaning these are words that are used in the language with you cannot use as a variable and so on.

So, you have module, end module and in this case, we have designing a circuit which is 2 is to 1 multiplexer. So, I am calling that as multiplexer. so module followed by module name, it is multiplexer and what are the connections to multiplexer that must be 3 inputs A B and select line; that is 2 to 1 marks and there is 1 output called out. So, there are four pins, if you buy a 2 to 1 marks, you will see four pins for the data and may be you will see power pins also.

So, I am we are not showing the power pins explicitly here, it is not necessary, you will only show the data input and output connection. And what you see here are called ports. So, ports are you have a circuit, what are the inputs and what are the outputs to it. Pins may include all your power supply, clock, whatever, whereas, ports are what are the inputs to this design and what are the output from the design.

So, we have four ports namely A B select and output and if I tell you this as the four ports, this node verilog by itself does not know, what does input and what does output. Verilog does not know, what multiplexer is; you have to explicitly say which are the inputs and which are the outputs. So, in this case, A B and select C l or inputs, so we indicate that by saying inputs A comma B comma cell and we want out to be the output port, we explicitly say output out.

So, far it is all clear, then what did we want, let say we want implement this circuit here. We need 2 AND gates, 1 OR gate, 1 inverter, we also need all these wires to connect intermediate. Once, for example, there is a wire that is coming out of here and there are two wires at coming out of the AND gate, which should go to the input of the OR gate, we need all of these connections.

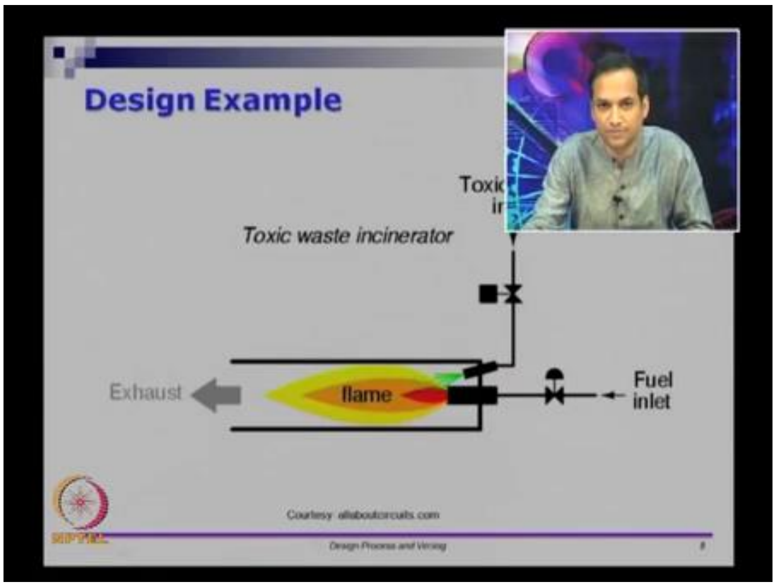
So, to do that, we need three wires, there are two wires temp 1, temp naught and one wire called select bar, which is going to be compliment of select. So, this is something I showed in video in week 1 itself. So, and temp 1 B self, this line is saying, I want a NAND gate which will take B and select as inputs and which will produce temp 1 as output.

So, if I think of it as a physical circuit, I am picking a NAND gate and to it is inputs, I am giving B and select it is output is called temp 1. Similarly, NOT of select this select bar and of A and select bar is temp 0 and OR of temp 0 and temp 1 is output; that is what this description says. So, in this description, we also have AND, NOT and OR here, these are mark in red, they are also keywords, they are called verilog primitives.

So, these primitives are therefore, Boolean functions, basic Boolean functions to be use directly. So, in this case, this primitive take 2 inputs and 1 output. So, remember all the primitives are only 1 output function. AND, OR, NOT, XOR all of these are actually a single output. So, the first thing will be for the output and rest of them will be for input. So, this one will imply that you have an AND gate with temp 1 as the output B and select as output. This one is OR has out as the output and temp 0 and temp 1 as the inputs, all the internal nodes are declared as wires.

So, the second set of lines is inputs and outputs, then you have the design description given here followed by end module. So, this is the very simplified version of verilog syntax.

(Refer Slide Time: 19:05)



Design Example

So, now, I am going to take a specific example and right verilog code for you. So, let us take this example. So, what we are having a let us assume that, there is a hospital in which there is medical waste, it could be tissue, it could be the gauze that the use are, the cloth that the use are, scrub are whatever. All these are toxic, you do not want this go to environment directly.

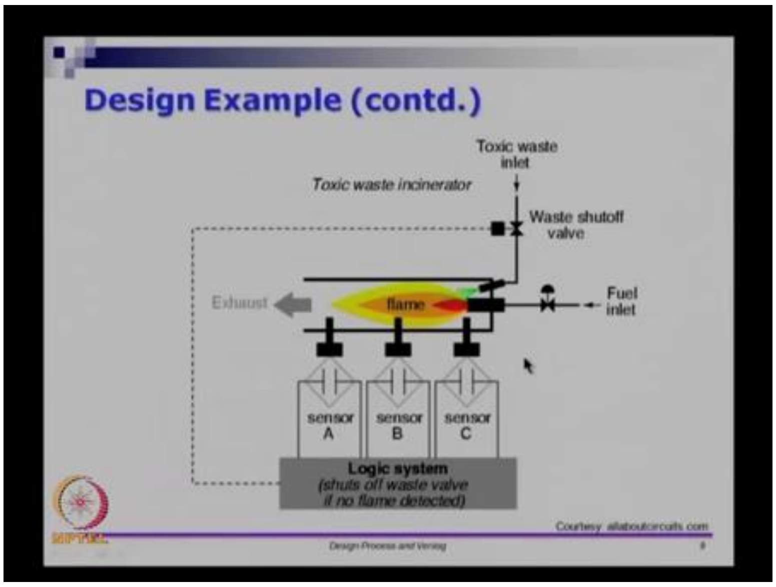
So, typically what hospital do is, the take the input and they burn, they take all these thinks and burn them, so anything that is coming out of the surgery. So, they may remove disuse, but in the process doctors may have used gloves and what not. They take all of that and sent it to incinerator, the incinerator is suppose to take them and burn them and that is now safe for taking out to the environment.

Otherwise, you may have all the bacteria and vireo that may go out of the environment, you do not want that. So, let say, you take this toxic waste as an inlet and there is a valve which controls the toxic waste. What we going to do is, you going to bring the waste into a chamber and we are going to inject fuel in to it and the fuel is suppose to burn the toxic inputs, toxic waste.

So, let us assume that the fuel can actually burn everything down and sent it out, when we sent it out, let assume that the output get sterilized. So, the issue is, we are having a fuel injection. So, there are various thinks that can happen. So, the fuel itself get is over, the fuel get over do not want keep pumping more and more inputs, because the input may go without getting burnt, it may go to the output, you do not that.

So, what we want is, you want have a mechanism by which we have to continuously monitor, whether there is a flame and for whatever the reason if the flame is not there, you want to immediately shut of this valve. So, that no more inputs are taken and we need to go and fix this set up before which we can start operating this one.

(Refer Slide Time: 21:05)



Design Example (contd.)

Toxic waste inlet
Toxic waste incinerator
Waste shutoff valve
Exhaust
flame
Fuel inlet
sensor A
sensor B
sensor C
Logic system (shuts off waste valve if no flame detected)
Courtesy allaboutcircuits.com
Design Process and Verilog
8

So, to do this just like in one of the earlier example, I have showed, I am going to put various sensors, let us assume that I put three sensors A, B and C. The job of this sensor is to figure out, if the flame is there are not. So, again like a set the flame could go of due to various reasons, maybe there is no fuel or there was power cut, it could be various things. So, you do not want a mechanical valve to continuously dump this medical waste without checking, what is happening with in this incinerator chamber.

So, I am going to put three sensor A B and C, there going to verify, whether there is flame or not as long, as there is a flame, as long there is flame things will get burn and the valve can remain open. But, if there is no flame, then I want to shut of this valve, to do that, I want a Boolean circuit or I want a digital circuit. So, we want to put a logic system that will shut of the ways valve, if the no flame is detected. So, this is a simple description problem.

So, this design problem is interesting, because we can show various interesting aspect within. So, let see this, what can happen is these three sensor, if they are always working and if there is a flame, they will give 1 1 1 as then an output, they will they will say that the flame is there. If the flame is not there and if the sensors are always working you will have 0 0 0 as the output.

However, what can happen is, the sensor themselves may go back, that the sensor also subject to failure, may be this sensor was replaced 5 months ago, these two are new sensors. So, the eventually this sensor fails, I may have to replaces this battery or I may have to replaces sensor itself, it could be a any of those. So, if I want something like that, then it possible that you have three sensors, but some of them are failing.

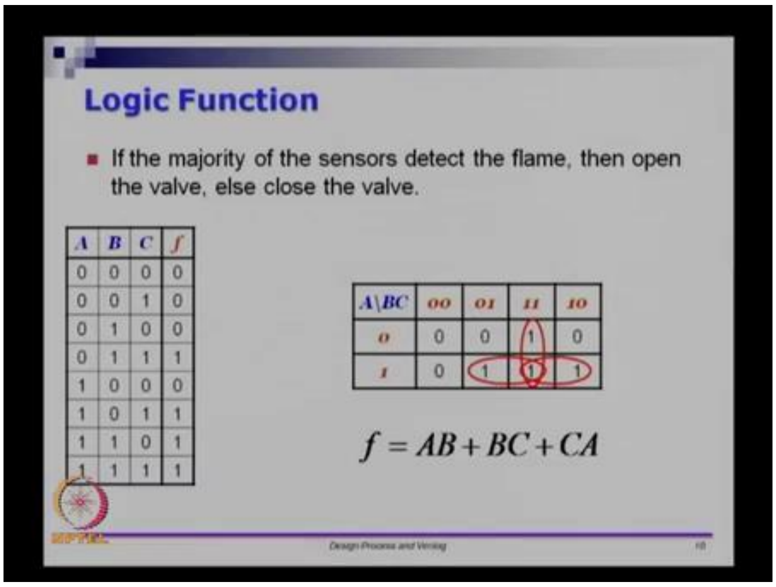
You want to build a logic system such a way that, even of this something which is failing I do not want an exhaust to ever have toxic waste automatically go down, the worst case have to shut it down and I have to be careful about this. So, to do this, one way in which you can achieve this is use, what is called majority function, I already talk about this in the previous module. So, if the majority of the sensors deduct the flame, then open the valve or close the valve.

So, this is the safe way in which you can operate, so it is not that you are looking for all three of them tell you that there is a flame. If there is a flame, if all the sensors are working all the time, then it simple to say, I will go and look at the all the sensors, if all of them are 1, then it is okay. But, what can happen is some sensors we know that, some sensors will fail, so if the sensor A B and C give 1 0 and 1 as output.

A and B are sensing of flame, A and c are sensing of flame, B is however not sensing of flame, what do we do now. So, what we want to B able to do is, to look at something like this, if the majority of the sensors deduct the flame, then open the valve, because it is possible that some sensors of fail. And they are not deducting any flame at all else close the valve.

So, we are assuming that, the valve then the failes, they will always say that the flame is not there. So, we do not have a condition, where something fails and by mistake it deducts of flame; let us assume that you do not have a sensor like that. The sensors, if there is no flame, they will say no flame, if there is a flame, they will say, there is a flame. However, then the sensors go back, it will give you only one think saying that, the flame is not there, it will not by mistake say is the flame is there.

(Refer Slide Time: 24:56)



Logic Function

- If the majority of the sensors detect the flame, then open the valve, else close the valve.

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

A\BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$f = AB + BC + CA$$

Design Process and Verilog

Let say that is the case, then the majority function will do that. So, if I look at the 3 inputs, all the three sensors say that there is no flame, then I can conclusion that, there is no flame, I have to shut of the valve. If all of them say 1, then the majority of them saying a 1, because all of them saying 1, so it is 1. Look at the other entries, let us look at this entry for example, 0 1 0; two sensors are not deducting the flame, one sensors are deducting in the flame, which means there is something wrong in this system.

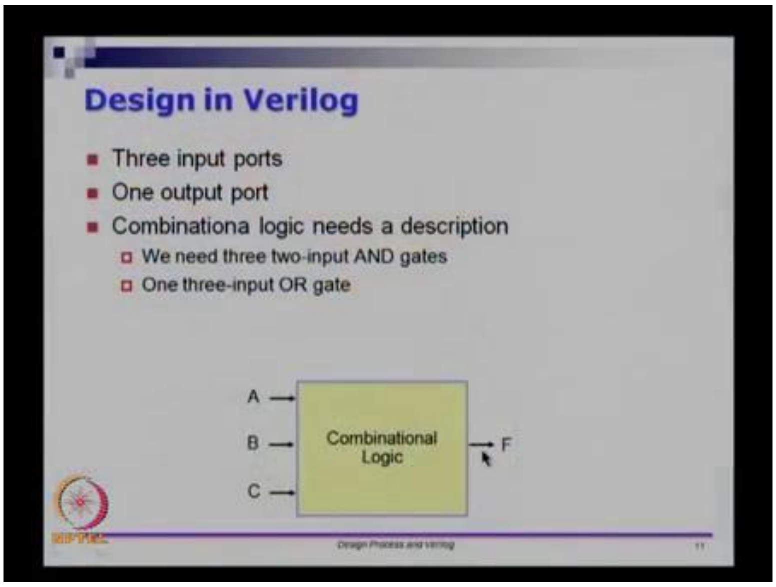
There are two sensors which are not deducting, one sensor is deducting. So, we do not know, whether there is a flame or not, that the majority says that, there is no flame. So, we will say that, there is no flame. Similarly, if two of them say, there is a flame and one of them says, there is a no flame, the majority says, there is a flame, so we will use 1. So, that is the truth table, return as a Karnaugh map, you have this entry.

So, you know how to write it, I will not going to the details now, I want to map these things. So, to map, there are three groupings of one that are possible. So, the vertical group of 1, these two can be group together and these two can be group together and I can go and write it as a Boolean function. So, f equals A B plus B C plus C A is the Boolean function, this takes the majority of the inputs.

You can see it here, if either A B or B C or C A is 1, immediately you can say f is 1, otherwise you have to look for the other variables also. So, this is the Boolean function for a majority 3. So, this majority of 3 inputs, this is useful in various cases, let say, you and your two friends want to go and have dinner and you are deciding, whether you want to go and have pizza or whether you want to go and have let say regular meals somewhere else, you want to take a vote.

So, you can take a vote and say, what over is a majority wins in. So, this is also something that can be captured by this function. So, you take individual votes and combine it using this A B plus B C plus C A, the majority function will tell you, what the majority of you have decided. Of course, all of you may decide, you want to go and have pizza or all of you may decide, you want to go to some other restaurant. So, that is the use for a majority function.

(Refer Slide Time: 27:24)



Design in Verilog

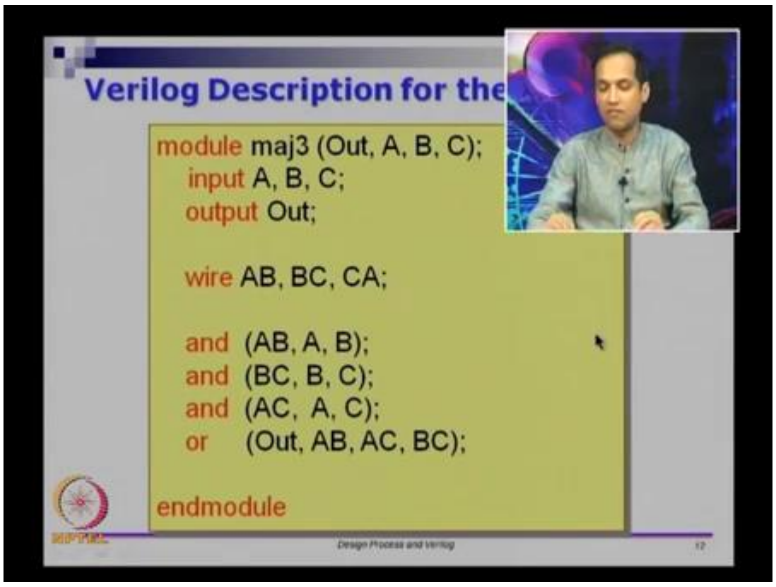
- Three input ports
- One output port
- Combinationa logic needs a description
  - We need three two-input AND gates
  - One three-input OR gate

A
B
C
Combinational Logic
F

Design Process and Verilog

Now, I want to go and do this in terms of verilog. So, I want to design a majority three circuit. So, it is a single output bit, this output will control, whether the valve is shut off or not, if f is 1, then there is flame, you do not have to shut of the valve, if f is 0, then we are going to the flame is off, we will go and shut off the valve. So, we need a logic description, this as 3 input ports and 1 output ports, it still need combinational logic here we already know that A B plus B C plus C A, we need 3 input, 2 input AND gates and 1, 3 input OR gate, so we need that.

(Refer Slide Time: 27:58)



Verilog Description for the

```
module maj3 (Out, A, B, C);
  input A, B, C;
  output Out;

  wire AB, BC, CA;

  and  (AB, A, B);
  and  (BC, B, C);
  and  (AC,  A, C);
  or   (Out, AB, AC, BC);

endmodule
```

Design Process and Verilog

Let see how to write this in verilog, so the first line I am going to write is keyword module and I am going to give a name to the circuit. So, any circuit that you design, you have to given a name in verilog, it is compulsory, it can be any name that you come up with. So, I am now calling this circuit as m a j 3, it is stands for majority 3, majority 3 has four ports out A, B and C; they are all comma separated and end of this, you see a semi colon, this is the interface description of this module.

As I said earlier, just the interface description is not enough, these are input or output ports, if you have say, whether this are input or output ports. So, in this case A, B and C are input ports and out is an output port. So, that explicitly describe in lines below that. So, we only say that there are four ports outside and in these two lines, we declare, whether there are in put or output.

In this example, there are 3 inputs and 1 output and I am also going to follow the convention whenever possible, who have the outputs before the list of inputs. So, in this case is the single outputs, I put it at the front and all the other things are inputs here. So, if I have more than 1 output, I pick some ordering of the outputs, I will put them first, and then I will put the inputs.

There is a reason, why I do that, I come to that later, then we needed three intermediate notes. So, the gate which does A B will need a wire, the gate which does B C will need a wire and the gate, which does C A will need a wire. So, we need three wires A B, B C and C A. So, these are the intermediate notes and now, I am ready to do how to gets these individual components.

So, how do I get A B, A B should be the AND of A and B. So, what this is actually saying is, I am going to pick a AND gate and to this AND gates, I am going to give one of the input as A, one of the input as B and the output from the AND gate, I can put a wire and say, I am going to call this A B, This is what you would do, if you do this in breadboard, you go and pick a AND gate and you will put it on the breadboard, you will connect two inputs A and B to it. And wherever the output pin is you will take that and you will take the dangling for a wire.

So, you know you want to use this later, you will keep it dangling, then similarly you need logical AND of B and C, as well as A and C. Now, there are three AND gates and the three output wires are dandling. So, remember wires are bidirectional, there is nothing. So, if you pick a wire, you cannot say this is the input and this is the output correct. So, wires are an bidirectional, the same applies to wires in verilog also.

So, there are three inputs A B C and 1 output out, these are also wire and then you have wires A B, B C and C A, which are all intermediate wires that you need. So, you have pick 3 AND gates and you have three dangling wires now you are called A B, B C and C A, they form the one end of the wire, the other end of the wire must go to the 3 input OR gate. So, this are out comma A B C, A B C is essentially doing this. So, this OR is a primitive, there is one output rest of this input.

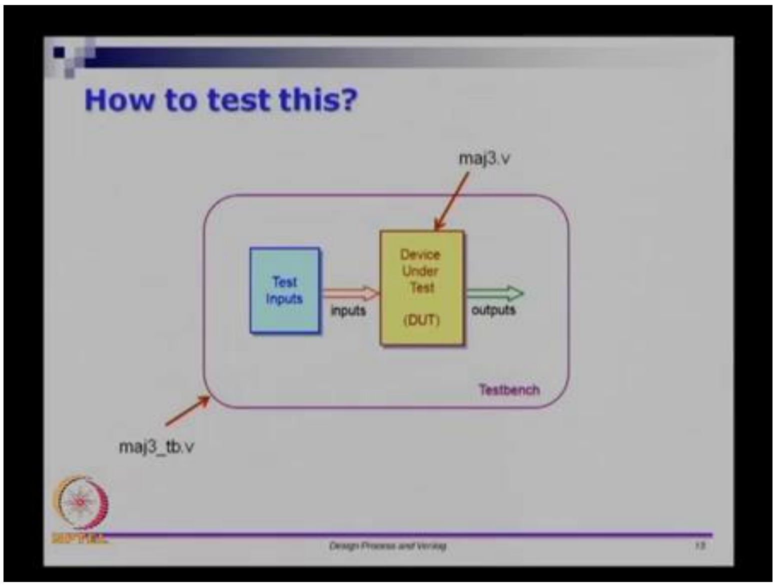
So, look at the AND gate is 1 output two of them are inputs, in this OR gate, there is 1 output in three of them are inputs. So, since all the primitives are single output function, the first one is always an output, the rest are inputs. It can be 2 inputs, 3 inputs, 4 inputs, 5 inputs; it can be any number of inputs for the primitives. In this case, I have used 2 input AND gate and I want to use a one single 3 input OR gate. So, this implies that there is a three input OR gates, this is the output called out and these are the three inputs that are the coming into the gate.

So, this gives as the description that I will take A B and C, I will give them to the three gates A B, B C and C A, there will be three wires that come out of it, these three wires will go to the 3 input OR gate, which gives you the output. We are done with the description of the circuit, so the last line is n module which finishes the description of the module.

So, you can see that by looking at this you can actually go and draw a circuit, if I gave you this, without telling you what the circuit is, if I gave you regular description, everything that you need to get a circuit is there. So, if I gave only this, you can see that, I am designing a black box which as 1 output and 3 inputs. In this black box, the 3 inputs are called A B C and the out output is called out. I will need three more wires inside and I will need four gates of this kind and this tells you, this is the connection that you need.

So, all of that is there, anything that you need to describe the circuit is there. So, given a verilog description, you can draw circuit. We start the other way round, I gave you the circuit and I ask for I wrote verilog, because we do not know verilog gate. But, given verilog later you may be able to understand, what the circuit is by just looking at it.

(Refer Slide Time: 31:15)



How to test this?

Now, let say design the circuit to do this, how do you test it in the lab, in the lab, you will put it inside, what is called test bench. So, what you will do is this, you probably pick up a breadboard, you will put the all these components and so on, connect them, you have the design ready. And once, you have it ready, you want to check, whether it is correct or not, may be you teacher wants you evaluate, you want to check yourself, whether the circuit is correct or not before getting it evaluated.

What will you do, you will go and connect the input wires that are there, may be to viridian ground to decide, whether it is 0 or 1. And at the output, you may either connected to oscilloscope or you may connected a LED and see, whether it is glows or not so on; that is a test bench. What you have is, you have a bench in some sense in which you put your design, your able to give inputs to it is by connecting to power supply.

You are able to take the output and see, what the output is either by looking at oscilloscope to which you have connected or by looking at LED’s or something like that to which you have connected. It could also be something like, I will go and drive the motor, the motor shoot run, it could be any of those. So, you need a test bed in which you can check this. So, we are going to call that a test bench, at test bench is going to take the device under test or DUT.

So, device under test in this example is majority 3, m a j 3 and what I am going do is, I am going to instantiate; I want to have one copy of this circuit to test it. If I want to check, whether it is works or not, I want to take one copy of the circuit which we are going to call instantiation. We instantiated, we need a mechanism by which we can give test inputs to it and we need a mechanism by which we can give we can observe the outputs.

So, from a software handle in some sense, you need to be able to take inputs to a program and outputs from the program, typically what it do is, you go and type the inputs in a keyboard or you go and type, you give a file as a input and so on. And the outputs could be something on a screen or something in a file; we need something like that for hardware description languages. We need to be able to give inputs and we need to be able to observe the outputs.

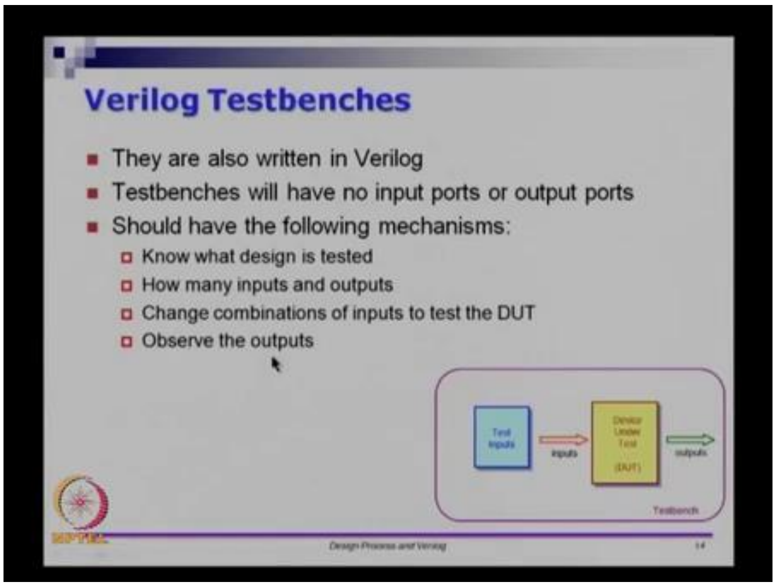
So, the output could be in terms of a file that gets printed or something that is printed on the screen or something you can see like a wave form and the inputs could be a mechanism by which you have give some specific inputs in some order. So, in a real circuit you may actually connect switches dips which to connect from 0 to 1 or so on, but if you going to write verilog, we need to write the whole thing inside verilog.

So, in verilog, what we going to do is, we are going to design verilog module called a test bench, this test bench is going to instantiate a module called device and the test. And in the test bench, we laid mechanism to provide it inputs and C D outputs or capture the outputs, we will say that output. So, what the usual practices the design that we have, we are going to write that in a file which ends with as dot v.

In this example, the device under test is majority 3 function, the module name is m a j 3, I am going to call that m a j 3 dot v. And the whole design itself, the test bench for it, I am going to write it in the file called majority 3 underscore t b, the t b will stands for test bench. So, m a j 3 underscore t b dot v is the test bench follows. So, the test bench if you look at it, it is not taking any external inputs, it is not giving any external outputs, it is generating all the signals that is needed for the device to be tested.

And whatever is coming out of the output, we are suppose to capture it is on over. The test bench itself is not taking any external inputs, NOR is it giving any external output. So, we will have to capture this in our description also. So, what I am going to show is how to write the verilog code and how to write the test bench for it in the next module. So, the verilog test benches are also return verilog test benches will have no inputs ports or output ports and it should have the following mechanisms.

(Refer Slide Time: 37:27)



Verilog Testbenches

- They are also written in Verilog
- Testbenches will have no input ports or output ports
- Should have the following mechanisms:
  - Know what design is tested
  - How many inputs and outputs
  - Change combinations of inputs to test the DUT
  - Observe the outputs

Test inputs
inputs
Device Under Test (DUT)
outputs
Testbench

Design Process and Verilog 14

You should know what designing you are testing, you should know how many inputs and outputs are there, you should be able to change the combinations of inputs to test the design and the test. And finally, if we able to observe the outputs, these are all the different things that we need. So, the next module is actually a demo of how to do this. So, just go back in the look at this slide, so far in this module observe the final details of the module and go and look inputs, outputs and all these other things.

Watch this video may be one more time, if you want to know what the different keywords and other things that we use so far are. So, in the next module, I am going to show you demo of this majority 3 function and we are going to do this online. And this is exactly what you will do for your home work and other things later.

So, thank you very much.