

Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering,
Indian Institute of Technology Ropar
Hardware Description Language: Verilog

(Refer Slide Time: 00:15)



Hardware description language

- Describe hardware
 - Functionality/behaviour
 - Delays and concurrency
- Why to model
 - Simulation and verification
 - Before chip design : Functional & Timing
 - After chip design : fault injection and diagnostic
 - Synthesis

Digital System Design 62

Hello again. In this part of lecture, we will see what are hardware description languages. Why they are used or in a very simple terms we will try to answer this question that why we are using Verilog in this particular course? So, before that let us try to answer even simpler question. Why do you use C, C plus plus, Java or Python kind of languages? So, you use these languages they are also called high level languages. Because, they are closer to English you can represent or you can write your ideas in a more precise manner in any of these high level languages like C, C plus plus, java, python, etc.

And whatever idea you are representing whatever idea you are writing that is very precise, that will have definitive input and a definitive output. Whatever you have written that will not give two different outputs unless it is aimed or it is designed like that. Similarly, so overall we are writing in those languages to express our algorithms to express our ideas which would be executed later on on a hardware, on a processor or an operating system in your computer systems.

Now, in a similar way instead of writing algorithms I want to describe my hardware. What does it what do we what do we mean by hardware? Hardware means that we will have these logic gates and we may have other hardware things like we may have chips or some modules

or some processors so you want to describe them. And to describe them we again one method what we have been learning in this course is we can describe them by Boolean equations.

Yes. Of course that is that is a fundamental method to describe these modules or these hardware units. However, when the hardware unit will become large or it will become will have many more components then this method will be insufficient. So, we would also require to represent these units hierarchically and so like one unit can have multiple other smaller units etcetera., plus there is a very definitive feature of these hardware units that all of them would work in parallel.

So, this functionality behavior part is one thing the other thing is because they have, they will work in parallel because all these units because it is a typical nature of hardware that, for example, you see you have a processor, keyboard, you see that all of them things looks like they are working in parallel. So, because hardware is working in parallel this particular property is called concurrency. So, we require a language we require some sort of description which can also have these characteristics of concurrency.

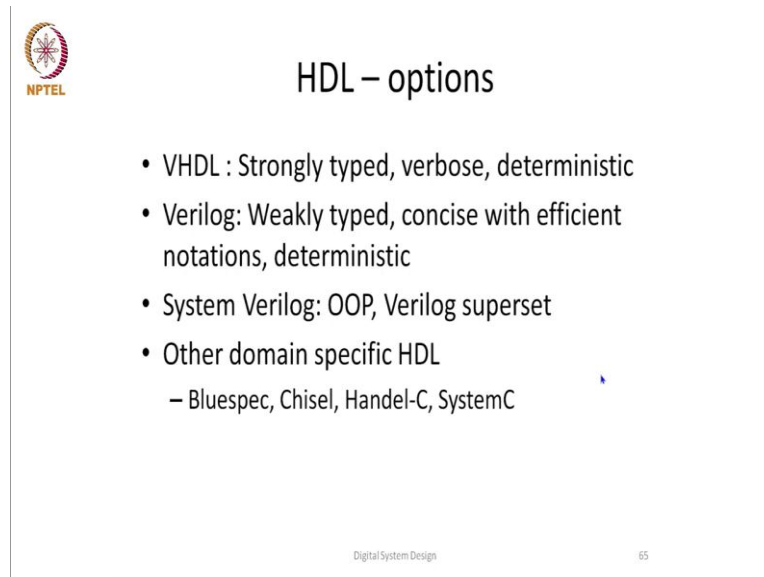
And they can also concurrency, whenever concurrency is there then the notion of delay will also be there, how much concurrent? So, whether there is a there is some delay between one function and the other function. So, we would require some sort of a language which can describe the function, behavior which can also model concurrency, which can also model delay so that is why a separate sort of languages were invented.

These languages are called hardware description languages. So, the languages which are specifically designed to describe the hardware. Initially, these hardware description languages like they were invented in early 80s. And then we got more or less confined in by the late 80s. So, many of these languages were defined so the idea of initial definition, initial invention of these languages what was to describe in hardware or to model in hardware.

So, when they were the whole idea was that before actually implementing things in hardware let us model them in software. So, that we can verify we can see whether it actually works or not. So, that was the initial idea of hardware description languages. But, in last 30 years like after late 90s, late 80s these hardware description languages are also popularly, popularly used to synthesize the hardware. Synthesizing hardware essentially means that once we write in hardware description languages then we can have some sort of EDA tool.

EDA means, electronic design automation tools. These EDA software, EDA tools are essentially software, so these EDA softwares can take hardware description languages. And can finally produce or finally describe the hardware in terms of logic gates and using those logic gates we can do the later process of chip designing. So, they are essentially now in summary we can say that these hardware description languages are also used to design hardware, not only model hardware but also used to design hardware.

(Refer Slide Time: 06:25)



The slide features the NPTEL logo in the top left corner. The title 'HDL - options' is centered at the top. Below the title is a bulleted list of HDL options. At the bottom of the slide, the text 'Digital System Design' and the number '65' are visible.

- VHDL : Strongly typed, verbose, deterministic
- Verilog: Weakly typed, concise with efficient notations, deterministic
- System Verilog: OOP, Verilog superset
- Other domain specific HDL
 - Bluespec, Chisel, Handel-C, SystemC

So, what are the various options of these hardware description languages? There are many present in the market, so there are two very popular languages one is VHDL, another is Verilog. So, VHDL is strongly typed verbose and deterministic. Strongly typed means that there would be like if there is one particular variable defined as particular data type then it would remain as the data type, so you cannot, so the type is very strongly defined, you cannot connect it to or you cannot pass this as a variable to another one which is slightly different. So, let us say unsigned int and int, they cannot be passed on as a single variable.

So, verbose means for the same thing you have to write a little more, deterministic, all the languages have to be deterministic that whatever function is being specified it will always behave in the same way. So, the result would always be deterministic if you run it once, twice, thrice it will always give the same result. The other popular language hardware description language is Verilog, it is weakly typed so and concise with efficient notations.

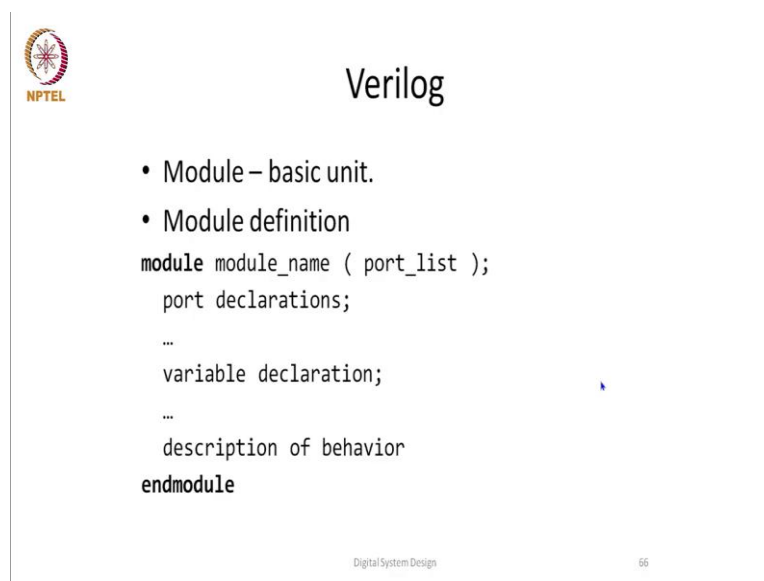
So, this particular language Verilog is mostly derived from C syntax. So, that is why it is quite concise. Weakly typed means it can help with a different kind of you can connect different kind of data types. Although, there would be some restrictions but yes, it is

less difficult to write in Verilog with respect to in VHDL. So, because of strongly typed because of verbose nature, historically VHDL has been used in academia and Verilog has been used in industry.

So, because of this particular reason we are also using because it is an industry oriented language, we would be using Verilog in this particular course. The other popular language is system Verilog. So, system Verilog is a Verilog superset, means whatever are the characteristics, whatever are the syntax in Verilog that will always be that you can always use in system Verilog. Along with that it can be used for a verification purpose.

So, there is some additional syntax which can be there in Verilog. So, system Verilog is a De facto standard for verification of designs. Along with this the other there are many other domain specific hardware description languages like Bluespec is a high level specification, even higher level than HDL. So, Chisel, Handel -C, SystemC, all of these are domain specific as well as even higher level of abstraction than HDL. So, let us focus on a Verilog hardware description language.

(Refer Slide Time: 09:27)

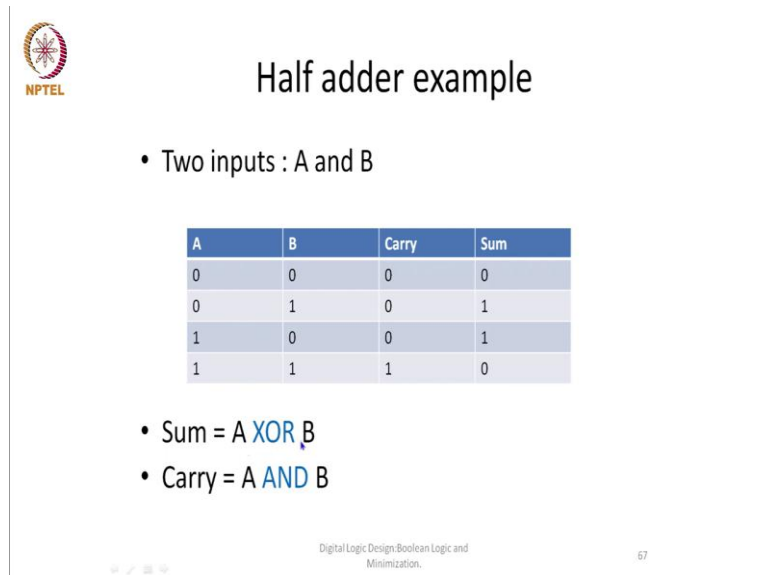


The slide features the NPTEL logo in the top left corner. The main title is 'Verilog'. Below the title, there are two bullet points: 'Module – basic unit.' and 'Module definition'. Under 'Module definition', the following Verilog code is shown: `module module_name (port_list);` followed by `port declarations;`, `...`, `variable declaration;`, `...`, `description of behavior`, and `endmodule`. At the bottom of the slide, the text 'Digital System Design' and the number '66' are visible.

In a Verilog hardware description language, the basic unit is a module. So, now this module is equivalent to any hardware unit. How do we signify a hardware unit? So, hardware unit will have some inputs some outputs. And those inputs and outputs will have certain name. So, these input and output in as a for a hardware it is also called ports. So, there would be some input ports there would be some output ports.

So, basic structure of a module or a hard Verilog module would be you will write module, then module name and then you will describe all the ports. Ports need to be declared, you need to tell that which particular port is a input port which particular port is the output port. Then you will describe the behavior of the module and then you can call end module. You can also treat this like a, you can also treat this like a C plus plus class or a C function. So, let us see how, let us take some example and then see that how it works.

(Refer Slide Time: 10:45)




The slide features the NPTEL logo in the top left corner. The title "Half adder example" is centered at the top. Below the title, a bullet point states "Two inputs : A and B". A truth table is presented with columns for A, B, Carry, and Sum. The rows show the combinations (0,0), (0,1), (1,0), and (1,1) with their corresponding Carry and Sum values. Below the table, two bullet points provide the logic equations: "Sum = A XOR B" and "Carry = A AND B". At the bottom, there is a footer with the text "Digital Logic Design: Boolean Logic and Minimization." and the number "67".

A	B	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

So, to take an example that I am taking an example of a half adder so there are two inputs A and B, there are two inputs A and B and the two outputs one is sum another is carry. So, when input is 0 and 0 sum is 0, carry is also 0. When input is A is 0 B is 1 then sum output is 1 and carry is 0. Similarly, if A is 1 B is 0 then also sum is 1 and carry is 0. If both A and B are 1 then sum is 0 and carry is 1. So, if we see from the truth table then a sum looks like a XOR gate and carry is an AND gate.

So, we can also write it like this sum is A XOR B and carry is A AND B. Now, we would like to define a hardware module or a Verilog module to describe this particular this particular functionality of half adder. And we will also see how we can verify this functionality. So, let us see how the Verilog module would look like.

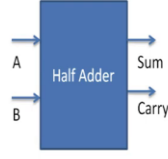
(Refer Slide Time: 12:04)



Verilog example – structural model

```
module half_adder(A, B, Sum, Carry);  
  
    input A, B;  
    output Sum, Carry;  
  
    xor U1(Sum, A, B);  
    // Sum = A XOR B;  
    and U2(Carry, A, B);  
    // Carry = A AND B  
  
endmodule;
```

Primitives that can be used
and, or, xor,
nand, nor, xnor
not



Digital System Design 68

So, in Verilog modules like while describing the hardware or describing our description of module there are two ways. One is a structural model another is a behavioral model. So, we will see both of these model, structural model as well as a behavioral model in this particular lecture. So, now this is my module a half adder which will have two inputs let us say A and B and two outputs sum and carry.

So, I would define my module as module half adder in the parenthesis I am writing all the ports A, B, sum and carry. And then I can also say then I need to describe what these ports are so I have to say this A and B are actually of type input and sum and carry are of type output. So, this is our module name and these are our port list. Good. So, now the description in structural model is you see we have to put them in the sort of, in list of connections between various components.

Now, in our previous slide we have seen that half adder in half at actually sum is an XOR of A and B and carry is AND of A and B. So, I can instantiate an XOR gate and then say that XOR gate and the name of instance I have to write, U1 is the name of instance and then the connections of this XOR. So you can also see it like this that XOR will also be some other module like this and it would have certain port list. So, if I write the name of my input and output in the same order whatever is the port list of that particular module then it would get connected.

So, this sum would be connected to the whatever is the output of XOR and A would be connected to first input of XOR, B would be connected to second input of the XOR and U1 is the instance name or the object name. So, you can see you can visualize that there could be


multiple XOR gates in our module. So, each of them should have different name so that is why this is the instance name. And similarly, for the AND gate we can use our another instance name and then connect the carry as the output, A and B as the input.

So, these AND or XOR, etc. So, these are all called primitive gates or primitive gates which are available in in our Verilog. So, we can use any of these primitive gates these are, there are six primitive gates which are there in in Verilog AND, OR, XOR, NAND, NOR and XNOR. Along with we have two buffer and NOT as the primitive gate which are used for a single input. All these primitive gates AND, OR, XOR, NAND, NOR and XNOR could be used and as a multiple input.

So, there could be a AND gate with multiple inputs. So, that is why the port pattern, this port pattern is first the output and after that all the inputs can be given. And this is true for all of these gates. Similarly, for the NOT gate first port would be output and then input. So, let us say after we have created this module half adder, now this half adder can also be used in some other module as a instance name. So, then we can say half adder and then module name and then we can connect. So this using this structural model you can create a hierarchy key.

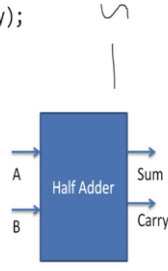
In hierarchy you can create different components and then then you can use those components in in higher level abstraction and keep on doing that till you can design your whole system. So, this structural module is a very powerful way of modelling a circuit, it is very similar to way we do on pen and paper that we create different models in terms of block diagram and each block diagram further can be designed using gates. And each gates could be could further be designed using these primitive gates. So, this is the way we do in a structural model.

(Refer Slide Time: 16:59)



Verilog example – dataflow model

```
module half_adder(A, B, Sum, Carry);  
  
    input A, B;  
    output Sum, Carry;  
  
    assign Sum = A ^ B;  
    // Sum = A XOR B;  
    assign Carry = A & B;  
    // Carry = A AND B  
  
endmodule;
```



Digital System Design 69

Another way of modelling the same thing would be in a data flow model. So, in a data flow model instead of specifying the structure means instead of specifying the gate level components we describe the behavior using Boolean equations. And all these Boolean equations, all the Boolean equations are used, use this particular keyword called assign. So, in this assign keyword we will write the Boolean equations in the same way we are writing in our while solving our questions etc.

The symbols what we have to use is for example for XOR we have to use this cap symbol for AND we have to use Ampersand and for NOT we have to use, for NOT we have to use this tilde symbol, for NOT we have to use this symbol and for OR we have to use this symbol. So, these symbols are used for, these symbols we can use for describing all the functions. And then these functions these Boolean equations can be assigned to the output.

So, the further this particular assign statement for example there are two assignment statements here, so both of them are happening concurrently. So, because it is a hardware model A and B, let us say there is a change in A and B both of the change in A and B would be reflected in sum as well as carry at the same time. So, this data flow model is also equally powerful like a structural model. Because, now instead of describing the whole basic level basic using instead of describing using basic gates now we can use equations.

Equations are sometimes more effective way of representing because you need not to rewrite each and every gate. So, you can you can summarize all of these gates in form of Boolean equations and then we can use them. And further because of you can use, you can combine multiple signals in some in sort of array, so those signals can be combined and then can be

assigned. So, this this all of these features make this data flow model also quite powerful. So, we will use this in some practice assignments and then we will understand. Now, let us see how do we verify a Verilog module.

(Refer Slide Time: 19:48)

The slide, titled "Verilog Testbench Example", features the NPTEL logo in the top left corner. It contains Verilog code for a testbench module named HA_TB. The code includes declarations for registers ra and rb, wires wsum and wcarry, and an instantiation of a half_adder module named ha_inst. The instantiation uses direct port connections: .A(ra), .B(rb), .Sum(wsum), and .Carry(wcarry). A diagram to the right of the code shows a block labeled "Half Adder" with two input ports, A and B, and two output ports, Sum and Carry. The testbench module HA_TB is shown with its inputs ra and rb connected to ports A and B, and its outputs wsum and wcarry connected to ports Sum and Carry. The slide footer includes "Digital System Design" and the number "70".

```
module HA_TB();  
    reg ra, rb;  
    wire wsum, wcarry;  
    half_adder ha_inst(.A(ra), .B(rb), .Sum(wsum), .Carry(wcarry));  
  
endmodule
```

To write, to verify a Verilog module, we also write another module this is called testbench. So, in this test bench we has a very specific property that it does not have any ports. Because, there is no input there is no output to the testbench. The whole idea this testbench is created is to instantiate our design under test or basically we call it duty. So, this half adder we want to test, so in the test bench we will instantiate some of the wires, some of the signals which will give input to, which will give input to these ports and which will also receive the output.

And we can then compare whether the output is correct or not so that is the functional functionality of our testbench. So, the other thing which test bench has to do, it has to instantiate our module which we want to test. So, here half adder is the module which you want to test. So, what we have to do is we have to instantiate the way we had instantiated AND gate and XOR gate in our structural design so we said that half adder is a name of module and the name of instance is h a underscore inst.


So, this is the instance of half adder. Now, in our previous example of structural modelling we have simply written sum A, B. Here we are doing a different kind of port binding. In the port binding we are specifying. So, here order is not important what we are saying that the A, A port is connected to ra signal of the test bench. So, you see ra is in the scope of test bench, so there is no dot written here but if you remember in our C or C plus plus whenever the scope is inside a particular module.

So, here half adder is the module and the scope of this A is half adder dot A. So, half adder is implicit here and dot A means if the scope of this dot A is inside a half adder. So, dot A is connected to ra of the of our testbench. Similarly, dot B is connected to rb of the testbench. Dot sum is connected to w sum and dot carry is connected to w carry. So, that is how we can make the connections and this is also called port binding.

So, this in this port binding we are connecting, so when we are using this particular style of port binding it does not matter what is the order of ports, order of variables within this instance. Now, you for defining or for connecting, for giving the input we have to define something ra and rb. So, we have to declare those signals or wires. So, the way these signals and wires has to be defined we have to, we are defining in in our testbench whatever is the input we will define it as a reg type, r e g reg type. And whatever is the output we are defining them as a wire. Whatever is the output that is defined as wire.

Now, you can question that why it, what is reg, what is wire? So, this reg and wire are a single bit, single bit structures which is very similar to bool of C, C plus plus a bool of C plus plus. And the reg and wire, so they are because they are close to hardware we are defining wire as a one-bit structure which can carry a particular input or output. A reg is another single bit structure which can also hold the value. So, because ra is a value which would be given to A, so it need to hold that value, so that is why we are using a reg for ra. Now, let us see how do we do it inside.

(Refer Slide Time: 24:35)



Verilog Testbench Example

```
module HA_TB();
  reg ra, rb;
  wire wsum, wcarry;
  half_adder ha_inst(.A(ra), .B(rb), .Sum(wsum), .Carry(wcarry));

  initial
  begin
    ra = 1'b0; rb = 1'b0;
    #10
    $display("A: %b, B: %b, Sum: %b, Carry:%b", ra, rb, wsum, wcarry);
    ra = 1'b1; rb = 1'b0;
    #10
    $display("A: %b, B: %b, Sum: %b, Carry:%b", ra, rb, wsum, wcarry);
    ra = 1'b0; rb = 1'b1;
    #10
    $display("A: %b, B: %b, Sum: %b, Carry:%b", ra, rb, wsum, wcarry);
    ra = 1'b1; rb = 1'b1;
    #10
    $display("A: %b, B: %b, Sum: %b, Carry:%b", ra, rb, wsum, wcarry);
  end
endmodule
```

Digital System Design 71

So, now we have to instantiate we have to give some particular input to our input ra and rb. So, this ra and rb all these inputs have to be given inside a initial block. So, in the initial there has to be begin there has to be an end. So, in an initial block so this begin and end are like starting curly braces and ending curly braces. And the, this particular module, this particular block is called initial. So, initial essentially means that it would be it would run first time or once whenever you are your execution will start.

Now, I want to assign ra as 0 and rb as also 0. So, what is this 0 0 means 1 bit, so since in Verilog you can also give input in in multiple bits so we have to define that how many bits it is. So, and you can also define give your input in form of binary, in form of hex, in terms of decimal, so that is why also it is important to specify whether it is binary, decimal or hexadecimal to be deterministic and to be precise.

So, because ra is a single bit so I am writing 1 and a dash b means that it is written in binary. So, this is the notion this is the representation this is the way we would be writing any constant input or we will be assigning these constant inputs to any variable like reg or wire. So, you will see that we have assigned 0 to ra, 0 to b, now after some time so this hash 10 means there is a delay element, so after 10 units of time, these is no unit specified so that means there is some unit which would be taken by Verilog by default.

So, after 10 units of time then we would give ra, we will give another stimulus or another input to a ra and rb. So, similarly after each 10 units of time we are giving different inputs to our ra and rb which are our inputs. And then the question would be, how do we see what is

the output? So, for seeing the output one particular method is we can use print f kind of a structures statements. So, the print f in Verilog is written as a as a display.

So, it is called dollar display. So, dollar display and after that the syntax is very similar to our C, so you can say inverted commas A and then percentage B that means the output would be written in binary form and output would be written in binary form. So, it is very similar to that and whatever are the variable names, so all these variable name you can write in the correct order. So, this display would give the output, give this particular string it will give and corresponding output would be generated. So, this whole thing is called stimulus and our display will give us the output or print f like output.