

Hardware Modeling using Verilog
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 06
Verilog Language Features (Part 1)

So, we now start our discussion where will be talking about some of the features of the Verilog language. So, the title of this lecture is titled Verilog language features; part one. Well, talking about Verilog, you have already seen some examples of sample Verilog programs. So, what we had seen is that the essential component of a Verilog program or Verilog code is a module just like in a high level language like C; you have function as the basic building block. So, the program will consist of one or more functions. So, in a Verilog program or code implementation, it will consist of one or more modules.

(Refer Slide Time: 01:12)

Concept of Verilog “Module”

- In Verilog, the basic unit of hardware is called a *module*.
 - A module cannot contain definition of other modules.
 - A module can, however, be *instantiated* within another module.
 - Instantiation allows the creation of a *hierarchy* in Verilog description.

```
module module_name (list_of_ports);  
  input/output declarations  
  local net declarations  
  Parallel statements  
endmodule
```

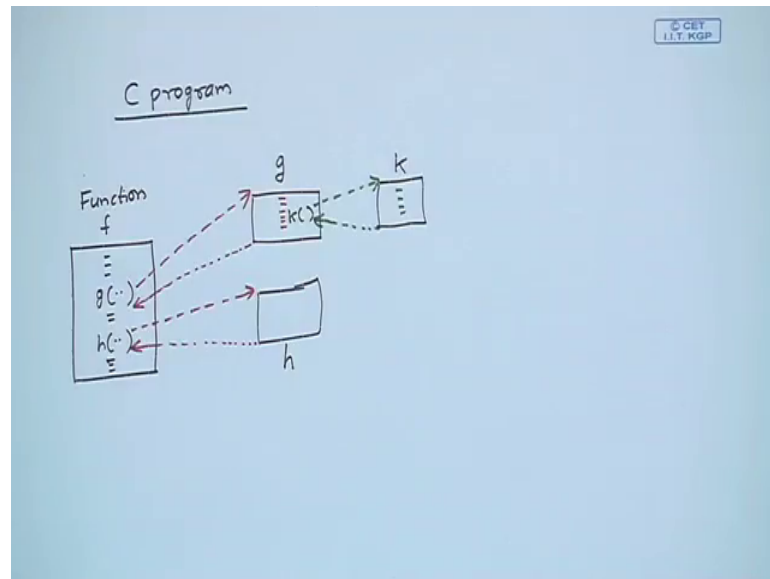
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, the concept of Verilog module is important. So, as I had said the basic building block or basic unit of hardware, in Verilog is called a module there are some restrictions like a module cannot contain definitions of other module, well, this is some similarity with a language like C. Let us say in C, whenever you define a function, you cannot define another function inside that function. So, the function definitions have to be disjoint. So, in a similar way, here also the module definitions have to be disjoint this is one. So, a

module cannot contain definition of other modules a module can be instantiated within another module.

So, here we have a mark difference between how a program like C works and how Verilog works.

(Refer Slide Time: 02:29)

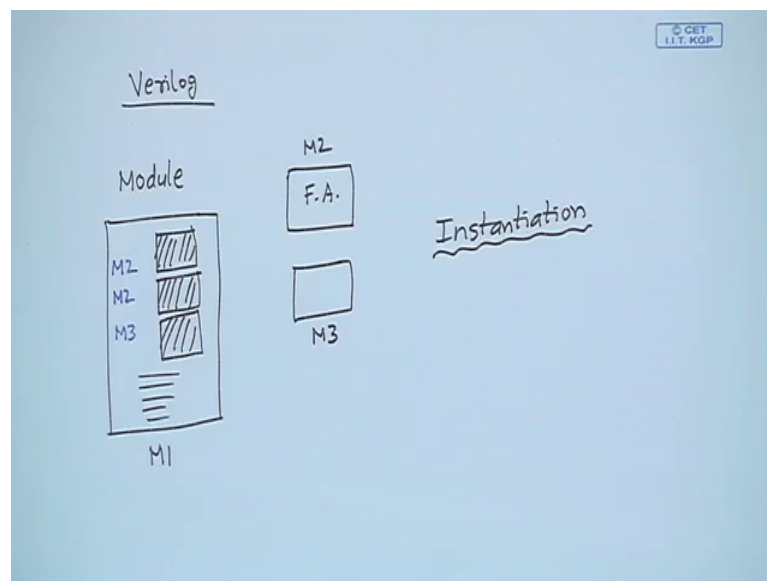


So, let me try to just explain well in a C program, let us say if I talk of a high level language like C, let us say there is a function f, this is the body of the function, there is a function g, there is another function h. So, within the body of the function, I can call g with parameters, I can call h and so on. So, in a typical program execution, whenever function is called, what happens; that when this function call is encountered, control will be transferred to this function. The code of this function will be executed and then control will be returned back to this statement following this g.

Similarly, when this h is encountered, again control will be transferred to this function h, it will be finished and then again it will return back. Now this can happen for nested call also, suppose there is another function. Let us say k where from inside g you have called k, right. So, when you call k from inside g, again a similar thing will happen k will be called, it will be executed and then it will be returned. So, if you look at the thread of execution during execution you encounter g, you go to g you execute encounter k you go to k finish it then come back then come back.

So, this is like a last in first out the last function to be called will be the first one to return right. So, in C program irrespective of how many times you call the function your size of the total program does not increase, just you are simply calling one function from another function, right. So, your program code is not increasing in that sense, but in Verilog the concept is slightly different because here we are talking about hardware. So, in Verilog, we have the concept of modules.

(Refer Slide Time: 05:00)

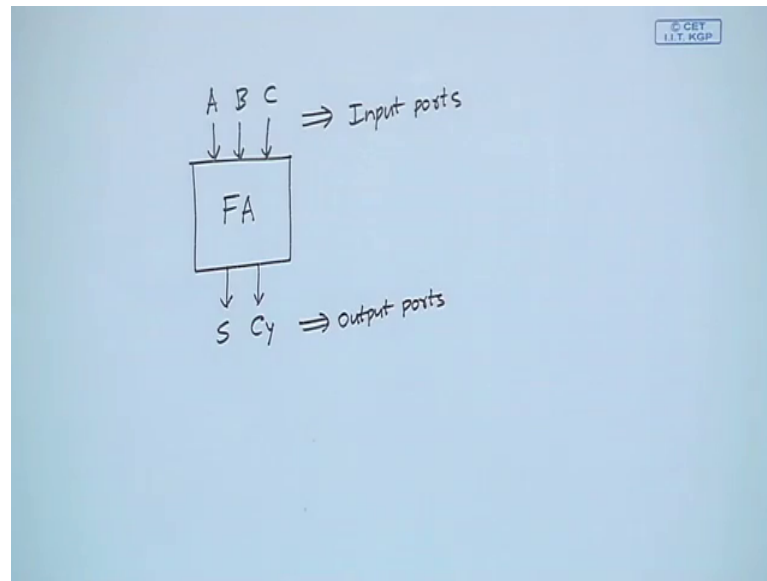


Let us say I have a module here, let us call this module is M 1. So, what I do from inside this module well I am invoking, let us say just a following the terminology of C, I am calling a module M 2, I am calling it 2 times, I am calling another module M 3 let us see. So, my M 2 module is already defined, my M 3 module is already defined. Now what will happen here whenever these modules are invoked. So, it is not exactly like calling a function in C like here for example, you are invoking M 2 2 times. So, what will happen inside this module, 2 copies of M 2 will get embedded. Suppose let us say, this is a full adder. So, if you call it 2 times. So, 2 copies of full adders will be put inside this master module, let us say M 3 is some other module you call it, this M 3 will also be embedded. Now in addition to these, this module M 1 can contain other statements also. So, this process of calling a module and the module getting inserted in the design in the hardware this is called instantiation.

Because we are talking about hardware there is no concept of calling hardware and returning well if you call it 2 times it means you are saying that I want 2 copies of that hardware. So, 2 copies are instantiated. So, M 3 is called once means I need one copy of M 3 just like that. So, recall the example that we showed you earlier of a 4 bit ripple carry adder there we had instantiated a full adder 4 times. So, that 4 copies of those full adders were included means in our ripple carry adder design fine. So, this is what is mentioned. So, a module can be instantiated within another module and this instantiation process this allows the creation of a hierarchy, let again talking about the ripple carry adder example you see ripple carry address the highest level as I said will consist of 4 full adders.

Now every full adder will consist of a sum and carry circuitry and every sum and carry circuitry if we implement using a structural description it will consist of some gates. So, this is a hierarchical description. So, the top level, we are seeing a full just an adder 4 bit adder next level we are seeing 4 full adders next level, we are seeing 4 carry circuitry and 4 sum circuitry and in the lowest level the gates right. So, a Verilog module in terms of its syntax contains the key word module to start with and it ends with end module and after module will come the module name just like just like in C, you have a function named module name and within parenthesis list of parameters, these are actually technically called ports and at the end there is a semicolon this has to be given now this is ultimately some hardware we are trying to design, let us say I am building a harder if this is a full adder.

(Refer Slide Time: 09:11)



So, you know that for a full adder there are 3 inputs A, B, C, this is a full adder and there are 2 outputs sum and carry out. So, this A, B, C, S and C y, these are called ports these 3 are so called input ports and these 2 are so called output ports.

So, in the declaration first you have to declare all the input output ports then inside your design you may be requiring some temporary connection like in the gate level net list this sum and the carry the description that was shown some of them you recall there some temporary signal lines wire were used, those are these local nets and after that there will be some statements now here we are calling them as parallel statements, well why we will call them parallel you see in Verilog, there are certain constructs which you will be learning one by one.

Now you see ultimately we are describing some hardware we are describing one piece of hardware say h 1, another hardware h 2, those are all part of the module now in terms of its operation h 1 and h 2 are both working in parallel because they are actually circuits they will always be fed with some inputs and they will be generating some output. So, it is not that you cannot say each one that well you wait till h 2 is complete then only you start. So, you cannot tell that to hardware like that because ultimately it is a circuit and circuit will respond whenever some inputs are changing, right. So, these are so called parallel statements which you shall see later in some more detail.

(Refer Slide Time: 11:17)

```
// A simple AND function
module simpleand (f, x, y);
  input x, y;
  output f;
  assign f = x & y;
endmodule
```

This is a behavioral description. The synthesis tool will decide how to realize f:

- Using a single AND gate
- Using a NAND gate followed by a NOT gate.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Let us take a simplest of simple example just an AND function we have already seen examples like this earlier using the assign statement. So, we have this module description name of the module is simple AND and the port list is f x and y and you specify x and y as inputs input is a keyword output is another keyword f is output and assigned f equal to x this is logical and well if you just mention x y and f it will mean these are all one bit variables; that means, they are logic signals. So, when I say ampersand. So, logically speaking you may see that well I am talking about an and gate, but well I should say, we are talking about an and function not necessarily an and gate because in this case.

So, you are specifying the behavior you are saying that the output will be the and function of x and y, but when you give this description to the synthesis tool synthesis tool will be actually generating the hardware now synthesis tool if it wants it can generate a single and gate, but alternately it can generate a NAND gate followed by a NOT; that is also and you see this second one may be more you can say more probable because in Cmos technology, it is much easier to design and implement NAND nor and not gates rather than AND an OR gates. So, this depends on the synthesis story. So, you really do not know for sure that what will be your final gate net list that this synthesizer will be generating.

(Refer Slide Time: 13:15)

```
/* A 2-level combinational circuit */
module two_level (a, b, c, d, f);
  input a, b, c, d;
  output f;
  wire t1, t2; // Intermediate lines
  assign t1 = a & b;
  assign t2 = ~(c | d);
  assign f = ~(t1 & t2);
endmodule
```

This is also behavioral description.

- One possible gate level realization is shown.
- t1 and t2 are intermediate lines; termed as wire data type.

The diagram shows a logic circuit with three gates: G1 (AND), G2 (NOR), and G3 (NAND). Inputs a and b are connected to G1, which produces output t1. Inputs c and d are connected to G2, which produces output t2. The outputs t1 and t2 are connected to G3, which produces the final output f.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

So, let us take another example, this is a 2 level combinational circuit. So, there 5 parameters the module name is 2 level. So, A, B, C, D are inputs output is f, you see here we have declared some intermediate nets t 1 and t 2 and there are 3 assign statements t 1 is doing an AND t 2 is doing a OR and then not basically not and f 1 is doing AND and not basically NAND. So, let us show the natural implementation t 1 equal to a and b suppose we implement it using a NAND gate t 2 equal to NOR of C and D, we implement using we using an nor gate and finally, f we implement using a NAND gate. So, this is a 2 level design and using assign, we have defined it in a some kind of a structure because we have also mentioned the interconnection this t 1 we have used here we have used here on the right hand side t 2, you have assigned t 2, we have used on the right hand side. So, this intermediate wires t 1 and t 2 they are serving the purpose of connecting these 3 gate outputs t 1 t 2 and finally, this f.

This is again just one possible gate realization because the synthesis tool can generate some other realization also for the same function. So, just one thing you remember that whenever you are using assign statement for describing some design, well, just one thing let me tell you the assign statement for most practical purposes are used to specify only combinational circuits, but later on I shall give you an example where assign statement can also be used to specify a sequential circuit this we shall see later, right. So, whenever you are using assign statements one thing you remember that you are actually talking about a behavioral description, right.

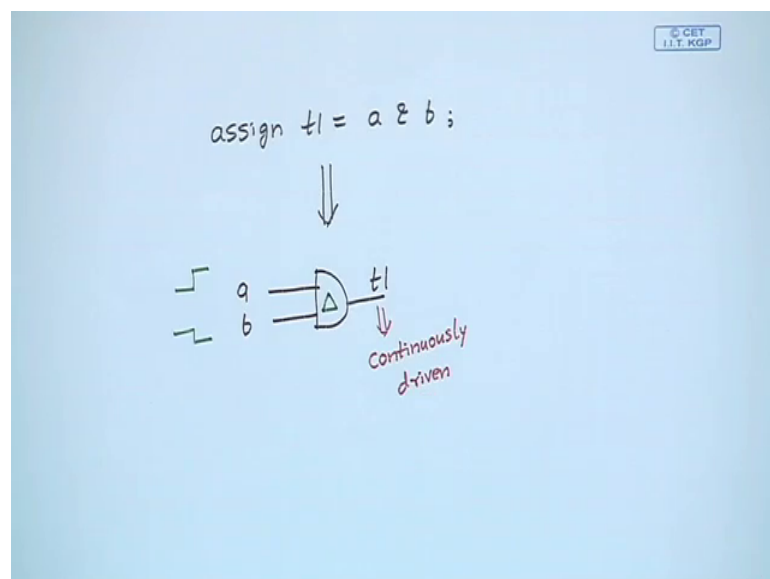
(Refer Slide Time: 15:41)

- Point to note:
 - The “assign” statement represents continuous assignment, whereby the variable on the LHS gets updated whenever the expression on the RHS changes.
$$\text{assign variable} = \text{expression};$$
 - The LHS must be a “net” type variable, typically a “wire”.
 - The RHS can contain both “register” and “net” type variables.
 - A Verilog module can contain any number of “assign” statements; they are typically placed in the beginning after the port declarations.
 - The “assign” statement models behavioral design style, and is typically used to model combinational circuits.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now, there are some points to note for the assign statements first it is that the assign statement represents something called continuous assignment. So, what you mean by continuous assignment where the variable on the left hand side. So, let us say I have an expression like this assign some variable equal to some expression let us say in a previous example assign t1 equal to A ampersand B, this is the expression. So, the variable on the left hand side will get updated whenever the expression on the right hand side changes.

(Refer Slide Time: 16:26)



So, what I mean is that when I write let us say assign t 1 equal to let us say a and b. So, possibly this will be mapped in to a hardware like this. So, we say this output signal t 1 this is continuously driven. So, what is the meaning of this continuously driven means whenever this input signal a and b changes, let us say a changes from 0 to 1 or b changes from 1 to 0, some change is there. So, whenever the input changes that will cause an immediate change in the output value depending on the functionality of the gate of course, there will be a delay of the gate after the delay this will change. So, there is nothing that says that well all the changes have to be synchronized with a clock. So, when a clock comes only then you change the output nothing like that the output signals t 1 is continuously being driven by the functionality of the gate the output of the gate whenever the input of the gate changes the output will also change and immediately the value of t 1 will change this is the idea continuously driven.

So, some constraints are there, well of course, net and register type variables, we shall be studying later. So, just assume for the time being that there are 2 kinds of variables net and register and wire which were used this is an example of a net type variables. So, this constraint says that this left hand side; this variable must always be a net type variable, but the right hand side can be either a register type or a net type left hand side cannot be register, this is the constraint now as the previous example shows. So, a module can contain any number of assignment statements this assign statements and as a matter of convention this assign statements are placed towards the beginning of the code right after the declaration of the ports. So, after your port line declarations and the temporary variables wires you place this assign statements after that and this is what I have said repeatedly that assign models the behavioral design style and is typically used to model combinational circuits, but you can also model sequential circuits using assign this we shall see later.

(Refer Slide Time: 19:47)

Data Types in Verilog

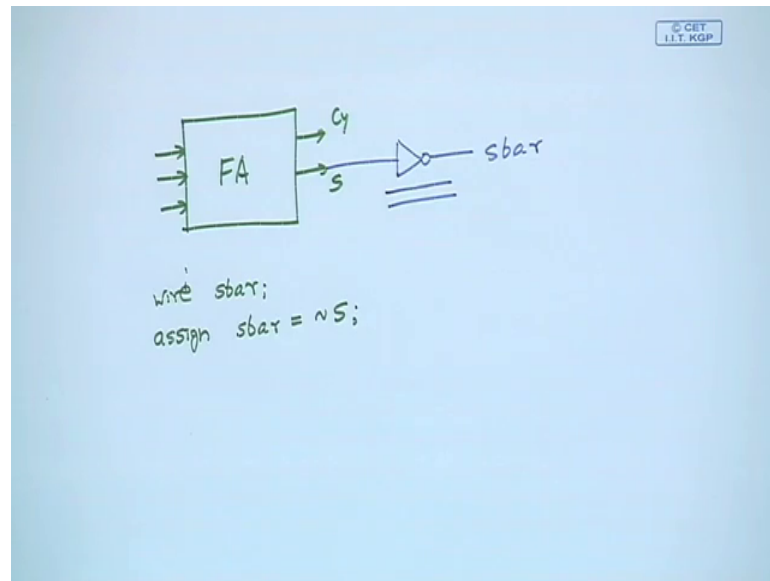
- A variable in Verilog belongs to one of two data types:
 - a) Net
 - Must be continuously driven.
 - Cannot be used to store a value.
 - Used to model connections between continuous assignments and instantiations.
 - b) Register
 - Retains the last value assigned to it.
 - Often used to represent storage elements, but sometimes it can translate to combinational circuits also.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Now, we have just now talked about net and register let us see what these are now. So, in Verilog, whenever you declare a variable just like in C whenever you declare a variable you declared it as either an integer float character and so on. So, in the same way whenever you declare a variable in Verilog you have to declare them with a particular given type. So, broadly speaking data types fall under 2 categories one is net other is register net just in the example that we took earlier sometime back net is continuously driven typically the output of a gate or output of a functional block that is connected to a net type variable. So, whenever that output of that block changes the variable that is connected to it, changes in a continuous way that is why you call it continuous assignment.

But we cannot use a net to store a value well whenever it changes you assign something again in the next time when you see its value will depend on the input of that gate or that block. So, it will not hold the value for a long time just like a flip flop or a latch you cannot store a value for a longer duration typically, a net is used to model connections between continuous assignments and instantiations what do mean by here.

(Refer Slide Time: 21:14)



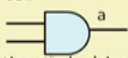
Let us say suppose I have done an instantiation in a module let us say. So, there is a full adder I have instantiated. So, there are 3 inputs and 2 outputs this is C_y and this is S . So, what I say in my module, I have written somewhere that assign let us say $sbar$ equal to naught of S where I have defined $sbar$ as wire. So, actually what this will do this will take the output from this instantiated full adder and this assign statement will just be an inverter, it will be driving in $sbar$ and this assignment will be continuously driven whenever s changes $sbar$ will change immediately right this is the basic idea.

Similarly, the register type variable, you can assign a value and the register is supposed to retain or store the value you can use it again later now as the name register implies well you may be tempted to think that a register is just like a hardware register that is constructed using flip flops well do not confuse the 2. This is a register type variable in Verilog and when you talk about hardware registers that is actually the implementation flip flops. So, herein, even if you declare a variable of type register; so, it is often used to represent storage elements, all right.

(Refer Slide Time: 23:19)

(a) Net data type

- Nets represents connection between hardware elements.
- Nets are continuously driven by the outputs of the devices they are connected to.



– Net "a" is continuously driven by the output of the AND gate.

- Nets are 1-bit values by default unless they are declared explicitly as vectors.
 - Default value of a net is "z".

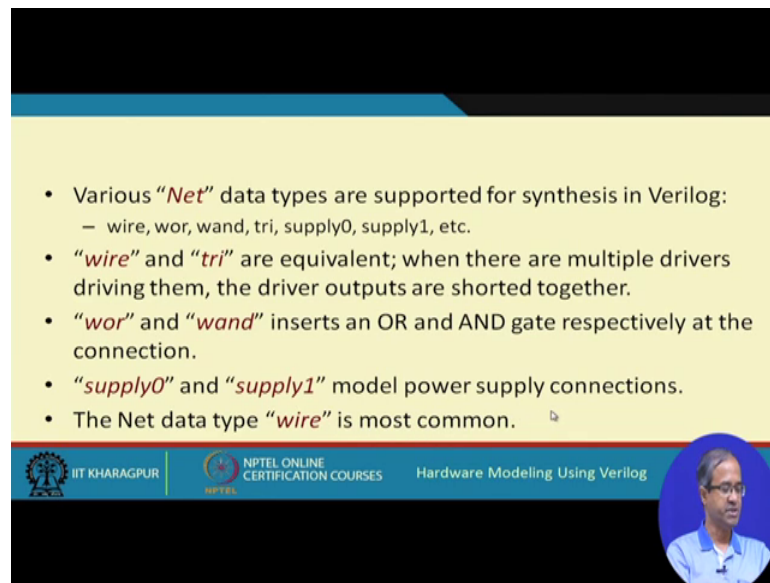
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

But there can be instances where a register can translate into combinational circuits also. So, this you have to remember now coming to the net data type we have already explained in some detail net actually represents connection between hardware elements and suppose this output, a is a net, this is continuously driven from the output of a gate they are continuously driven by the output of some device which they are connected to.

So, whenever the input of this gate changes this gate output will change and the value of a will also change immediately.

So, when you declare a net using say wire for example, say wire a. So, by default it will be taken as a 1 bit value, but we shall see that we can also declare vectors where instead of one bit value we can use a collection of lines like I can declare a variable to be a 4 bit variable 8 bit variable sixteen bit and so on. So, I can define something called a vector and use these vectors in my Verilog description or coding also this will see and one thing when you declare a net, but you have not initialized yet. So, by default the value is considered to be in the high impedance state z refers to high impedance or they tri state, right.

(Refer Slide Time: 24:49)



The slide features a yellow background with a blue header and footer. The main content is a bulleted list of Verilog net data types. At the bottom, there is a blue footer bar containing the IIT Khharagpur logo, NPTEL Online Certification Courses logo, and the course title 'Hardware Modeling Using Verilog'. A circular video feed of a male speaker is positioned in the bottom right corner of the slide.

- Various “*Net*” data types are supported for synthesis in Verilog:
 - wire, wor, wand, tri, supply0, supply1, etc.
- “*wire*” and “*tri*” are equivalent; when there are multiple drivers driving them, the driver outputs are shorted together.
- “*wor*” and “*wand*” inserts an OR and AND gate respectively at the connection.
- “*supply0*” and “*supply1*” model power supply connections.
- The Net data type “*wire*” is most common.

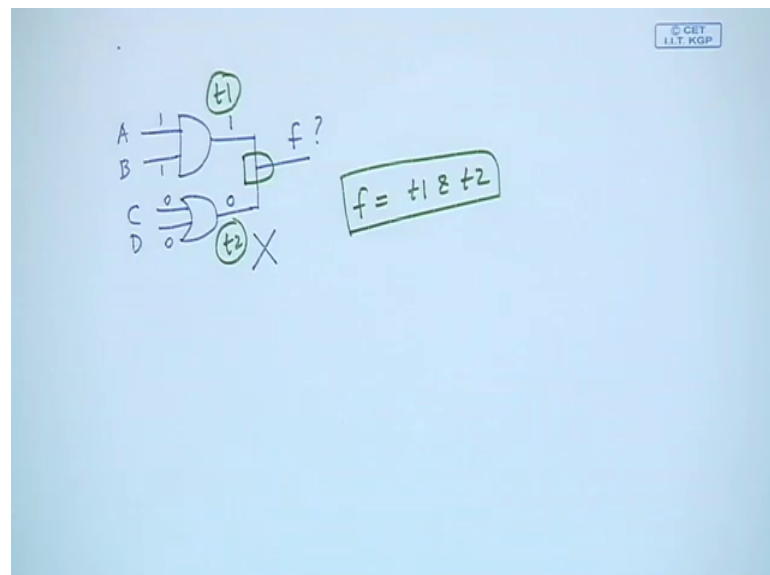
So, it means in Verilog, there are a large set of net data types that are supported some of the important ones I am showing here wire, this is wire or wire and tri state supply 0 and supply one, this wire and tri they are equivalent. So, when the multiple drivers driving them the driver outputs a short interval I will just show an example later this wired or and wire and they also outputs are shorted together, but there will be a OR and AND function at the junction, well, again I have an example, here I shall show and sometimes you may need to specify that which is your power supply line VDD and ground. So, there are 2 special data types supply 0 and supply one that indicate connections to ground and connection to VDD network wire of course, is most commonly used.

(Refer Slide Time: 25:57)

The slide compares two Verilog modules. The left module, 'use_wire', declares output 'f' as a 'wire' and uses two 'assign' statements: 'assign f = A & B;' and 'assign f = C | D;'. A note below states: 'For A = B = 1, and C = D = 0, f will be indeterminate.' The right module, 'use_wand', declares output 'f' as a 'wand' and uses the same two 'assign' statements. A note below states: 'Here, function realized will be f = (A & B) & (C | D)'. The slide footer includes IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and Hardware Modeling Using Verilog.

Let us take some examples here I have a module 5 ports are there, A, B, C, D are input output is f. Now here I am declaring f as type wire since output it, here I can specify the type of the output separately and you see I have declared or defined 2 assign statements both are assigning value to f; f equal to A and B f equal to C or A.

(Refer Slide Time: 26:38)

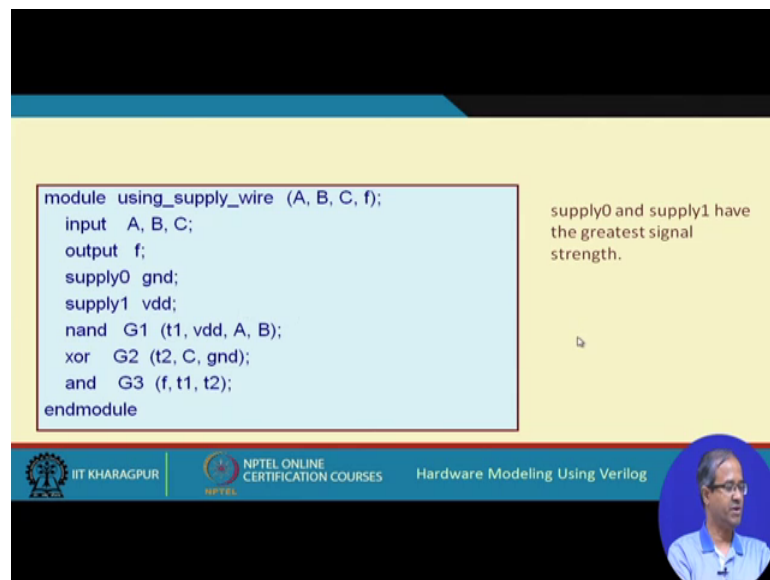


So, actually this is something one and gate which is A,B; there is another OR gate inputs are C and D both output are f. So, actually what you are saying that as if the outputs are shorted together and you are calling this f.

But you see this is not a normal practice and this is something which is prohibited in design. So, if I apply A 1, B 1 and C 0, B 0. So, one gate is trying to make the output one, other gate is trying to make the output 0. So, what should be f? It will be indeterminate, right. So, for certain condition the output f can be indetermined this is a wrong design you should not use this, but whenever you want to tie the outputs together you can use this either wired and or wired or you see instead of wire I use a wire and here what is wire and means meaning. So, wire and means that in this circuit there is an implied and gate here at the junction. So, so if you call this 2 lines as t 1 and t 2.

So, f will actually be t 1 and t 2. So, now, the output f is very well defined. So, if you declare output f as a wire and then an implicit and operation will be realized at the connection. So, if you declare it as wired or then a wire then here or operation will be done, these are all means operations or operators which have a direct correspondence with the hardware. So, in hardware we have this concept of wired AND and wired OR connections depending on the logic family, if you tie the output of 2 gates together for example, for C mos it works like a wired and if you connect the outputs of 2 C mos gates together the output will be 0 if any one of them is 0 that is how it works.


(Refer Slide Time: 28:44)



```
module using_supply_wire (A, B, C, f);
  input  A, B, C;
  output f;
  supply0 gnd;
  supply1 vdd;
  nand  G1 (t1, vdd, A, B);
  xor   G2 (t2, C, gnd);
  and   G3 (f, t1, t2);
endmodule
```

supply0 and supply1 have the greatest signal strength.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



So, let us take another example where we have also used supply lines supplies you will supply one; C supply one, I have defined a variable called VDD which is connected to supply one a variable called ground which is connected to supply 0 and we have

instantiated 3 gates NAND, XOR and here for some of the inputs I have used VDD on ground OR. So, this I can do if I want.

(Refer Slide Time: 29:18)

Data Values and Signal Strengths

- Verilog supports 4 value levels and 8 strength levels to model the functionality of real hardware.
 - Strength levels are typically used to resolve conflicts between signal drivers of different strengths in real circuits.

Value Level	Represents
0	Logic 0 state
1	Logic 1 state
x	Unknown logic state
z	High impedance state

Initialization:

- All unconnected nets are set to "z".
- All register variables set to "x".

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog

Just an illustration that how you can use the supply wires; so, data value and signal strength means Verilog supports, this 4 logic values 0 1 x and z 0 is logic, 0 1 is logic 1 x is unknown which is not initialized it can be 0 it can be 1, but z is high impedance state now when you initialize. So, all unconnected nets the wires they are initialized to z, but if you declare some variables of type register registers are initialized to x, well in addition to the values, this value levels there are strength levels also see strength level actually, they will talk about something like this if I connect 2 signals together.

So, which of the signal is stronger if my first signal is stronger, then the final output will be dominated by the stronger signal that is the concept of signal strength. So, in Verilog, there are 8 signal strengths which are defined while we should be talking about this later because this signal strength comes in to the picture particularly when you talk about mos transistors.


(Refer Slide Time: 30:25)

Strength	Type
supply	Driving
strong	Driving
pull	Driving
large	Storage
weak	Driving
medium	Storage
small	Storage
highz	High impedance

Strength increases

- If two signals of unequal strengths get driven on a wire, the stronger signal will prevail.
- These are particularly useful for MOS level circuits, e.g. dynamic MOS.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog



So, in circuits is the mos transistors, there are various points in the circuits where the signal strengths are different. So, if I connect 2 points together, their signal values are different strengths are different then the highest strength signal will dominate. So, just to know which one is higher which one is lower? So, this table just shows you that strength increases in this reaction. So, if 2 signals of unequal strengths are driven on a single wire, this stronger signal will prevail right. So, with this we come to the end of this lecture. So, we shall be continuing with our discussion in the next lecture as well.

Thank you.