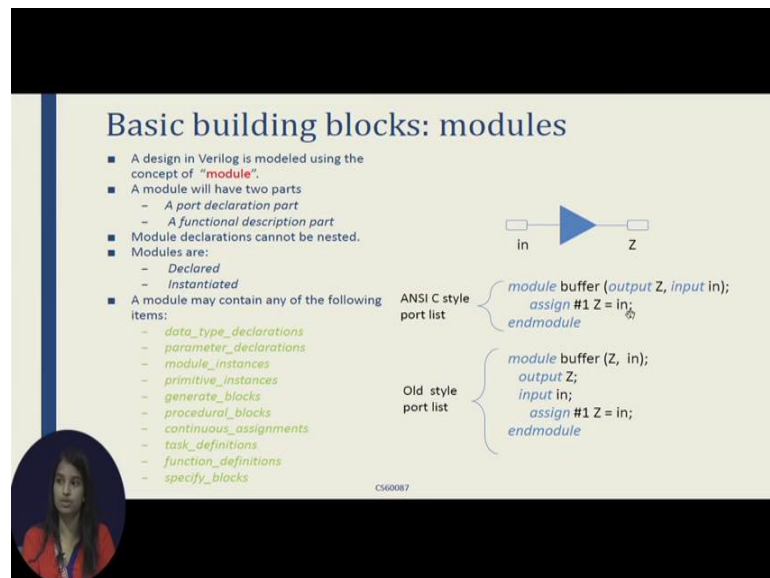


Embedded Systems Design
Prof. N. Vidya
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 10
Tutorial – II

Hello everyone. In the last tutorial we learned about HDL it is types, disadvantages and advantages in this tutorial we will learn about Verilog which is a type of HDL. So, let us get start with it.

(Refer Slide Time: 00:40)



Basic building blocks: modules

- A design in Verilog is modeled using the concept of "module".
- A module will have two parts
 - A port declaration part
 - A functional description part
- Module declarations cannot be nested.
- Modules are:
 - Declared
 - Instantiated
- A module may contain any of the following items:
 - data_type_declarations
 - parameter_declarations
 - module_instances
 - primitive_instances
 - generate_blocks
 - procedural_blocks
 - continuous_assignments
 - task_definitions
 - function_definitions
 - specify_blocks

ANSI C style port list

```
module buffer (output Z, input in);  
    assign #1 Z = in;  
endmodule
```

Old style port list

```
module buffer (Z, in);  
    output Z;  
    input in;  
    assign #1 Z = in;  
endmodule
```

The slide also includes a logic diagram of a buffer with input 'in' and output 'Z'.

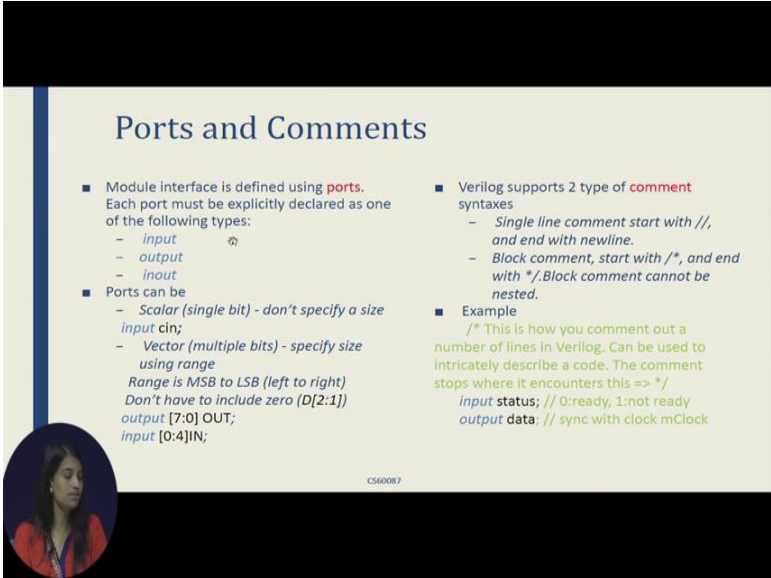
The basic building and we give a functionality of the design which tells us what the design does exactly. So, a module and n module are the key words that are used in Verilog and every design should have these key words. So, now the module declarations we cannot use like we cannot write module inside module, so we cannot use a nested module declaration, but we can use module inside a module; like we can declare another module inside a top model.

So, we will come to know how we instantiate it later slides. Currently you just need to know that modules can be declared and it can be instantiated in other module. So,

Verilog module in general may contain all these, means it may not contain some of these, but some of these are all always present. So, we will look at it one by one in the slides as we go on.

As you see there I have mentioned two types of codes here. So, the difference in here is Verilog was first developed by gateway automation design in 1995 it was standardized Verilog 1394. And it was then reviewed and again in 2001 there was edition there was up gradation of the standard. So, now the standard is merged into a system Verilog group, but the two slides that I have shown the first one is the new one and the second one is the older one. So, in the old design we used we had to mention the ports direction separately like output input separately, but in the new design we can put it together. Also each port has data types. So, in the older design we had to specify the data types separately. Here I have not shown because I wanted to make it simple we will know later what a data type and how ports and data types are related. So, this is the basic Verilog module.

(Refer Slide Time: 02:50)



Ports and Comments

- Module interface is defined using **ports**. Each port must be explicitly declared as one of the following types:
 - `input`
 - `output`
 - `inout`
- Ports can be
 - *Scalar (single bit) - don't specify a size*
`input cin;`
 - *Vector (multiple bits) - specify size using range*
Range is MSB to LSB (left to right)
Don't have to include zero (D[2:1])
`output [7:0] OUT;`
`input [0:4] IN;`
- Verilog supports 2 type of **comment** syntaxes
 - *Single line comment start with `//`, and end with newline.*
 - *Block comment, start with `/*`, and end with `*/`. Block comment cannot be nested.*
- Example
 - /* This is how you comment out a number of lines in Verilog. Can be used to intricately describe a code. The comment stops where it encounters this => `*/`*
 - `input status; // 0:ready, 1:not ready`
 - `output data; // sync with clock mClock`

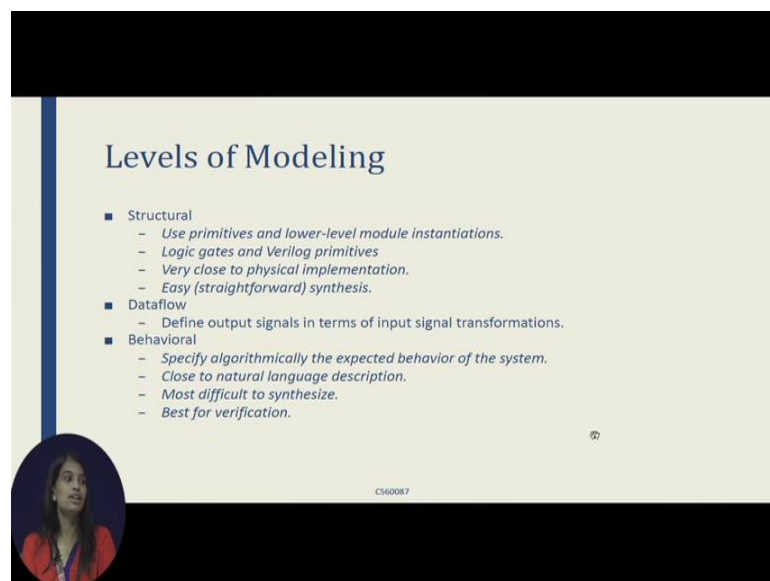
CS600087

So, next we will see what are codes and comments in a Verilog code. So, first as I discussed it earlier there can be three types: input ports, output ports and inout ports as per the design specification. And they can be in single bit ports and they can be multiple bit ports. So, we call them scalar and vector and you can see the examples here input and

the keyword and output and the keyword. And similarly we will have inout and some keyword. So, we can declare a single bit port or a multiple bit port. And in multiple bit port the range can vary from MSB to LSB left to right and it can take any value.

So, in Verilog we can mention 2 is to 1 and we can sorry we can mention 7 is to 0 or 0 is to 4. So, we will see later which kind of technique we use generally in practice, but this is possible. So, we can mention any type of a range in a vector. So, coming to the comments part of Verilog it is similar to C programming it has a single block single comment and a block comment and single line starts with double slash and block comment starts with slash star. So, this is an example of how we use a comment line in a Verilog. So, this whole is not executed at all, it is treated as comment and we can describe the design what a designer is going to do using a block comment or we can describe what a line does in a single line comment.

(Refer Slide Time: 04:39)



Levels of Modeling

- Structural
 - Use primitives and lower-level module instantiations.
 - Logic gates and Verilog primitives
 - Very close to physical implementation.
 - Easy (straightforward) synthesis.
- Dataflow
 - Define output signals in terms of input signal transformations.
- Behavioral
 - Specify algorithmically the expected behavior of the system.
 - Close to natural language description.
 - Most difficult to synthesize.
 - Best for verification.

CS60087

So, next we will learn about levels of modeling in Verilog. We have three basic type of levels in Verilog: structural, dataflow and behavioral. So, we will go through them one by one, but in the slide I just want you to know that there are three kinds of levels like the way in which we model a design. So, there are specific ways in which we model a design in Verilog.

In the structural module modeling we used primitives and low level module instantiation is similar to making a schematic. So, we know the basic building block of the design and we just instantiate the basic gates in the. So, it is very close to physical implementation and it is very easy to synthesize. So, it is more efficient if you compare with the dataflow and behavioral model. We will see in detail structural modeling.

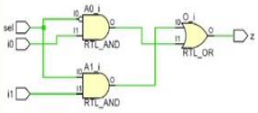
And in data modeling the input signal transformations are defined in terms of output; sorry output signal transformation output signals are defined in terms of input signal transformation. So, if there is any change in input will make the output look like the output is represented according to the input change. So, the main disadvantage of dataflow is we cannot model a memory element using a dataflow model.

Third is behavioral this is the most used model as of now it does not require for us to know the basic building block, we can specify the code in an algorithmic manner. And since it is close to a natural language description it is very difficult to synthesize because the tool does not know what we are asking the tool to design. So, this works best for verification purpose for testing the system, but we will see why we use this more. Like we used generally dataflow and behavioral mix in practice and we generally do not prefer structural modeling we will see how why we do not use it.

(Refer Slide Time: 06:54)

Structural Modeling

- Structural elements of a Verilog structural description are logic gates and user-defined components connected by wires
- Structural description can be viewed as a simple netlist composed of nets that connect instantiations of gates.
- Primitives are pre-defined gates already described in the Verilog language.
- Advantages:
 - Hand-Designed systems always the most efficient
 - Power, area, frequency requirement can be most accurately met by designer
 - Existence of proven sub-blocks
 - IP-Cores readily available
- Disadvantages
 - This kind of modeling can get tedious for very large designs
 - Architecture / hardware of problem being solved may not be known always
 - Arriving at structure first and then modeling it will take lots of time
 - Building edge triggered circuits with structural modeling alone can be very tricky
- Though being ideally the best way to model a hardware time-to-market issues and complexity involved works against it.



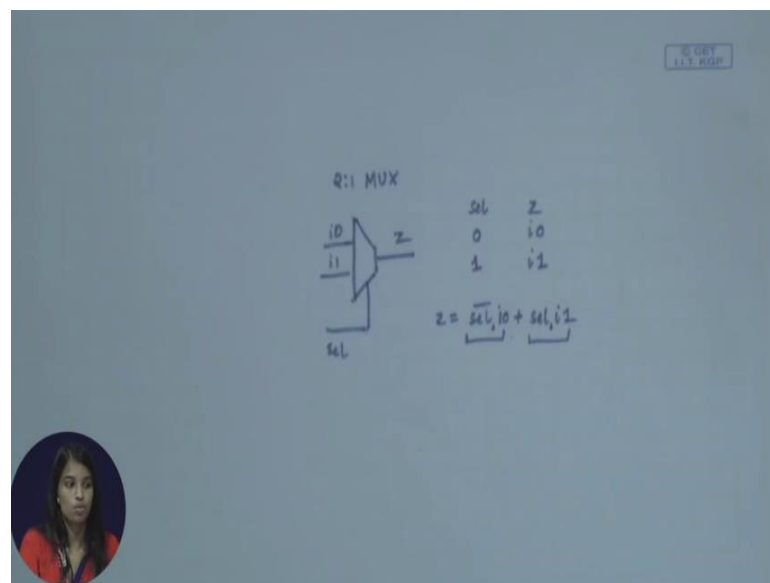
```

//2-to-1 Multiplexer
module mux2(
  output z,
  input i0, i1, sel );
  wire nsel,a0,a1;
  not inv (nsel, sel);
  and A0(a0, nsel, i0);
  and A1(a1, sel, i1);
  or O(z,a1,a0);
endmodule
            
```

CS60087

So, I will first discuss. So, what is structural modeling and how it is advantageous and what are the disadvantages of structural modeling. In structural modeling we use logic gates and user defined components connected using wires. So, as you can see in the slide I have taken an example of 2 to 1 multiplexer. So, 2 to 1 multiplexer looks something like this.

(Refer Slide Time: 07:28)



We have two inputs and an output and one select line. This is a 2 to 1 multiplexer and we have i 0 and i 1 as input and select line as select as input also and we have a z output. So, the module that I have defined is a MUX 2 is to 1 MUX. So, MUX 2; MUX 2 is a keyword. And I have a z as output i 0 as input i 1 as input and as select line.

So, the functionality of this MUX is defined in a way that when there is a change in select line; like when the select line is 0 we get an output of 0 i 0 and when the select line is 1 we get an output of i 1. So, this is the basic functionality that we need the MUX to operate like. So, if you know digital design concept this functionality can be represented using a Boolean equation. And the Boolean equation comes something like this z equal to select bar i 0 plus select i 1. So, if you see the code here in the slide we directly represent this Boolean equation in the form of logic gates.

So, we have one multiplication in the equation two ands in the equation and one or. So, we use two and gate one or gate and this is not as for the select not in the multiplexer design. So, in a structural modeling we used directly the logic gates to describe a design functionality and in Verilog language we have predefined gates like wire not and or not etcetera which need not be declared we can directly use it in the code. So, that is predefined in the Verilog language.

The advantage of this kind of modeling is that it is hand design. So, we know exactly what the design does and it is most efficient because we are actually thinking about the hardware, what is a hardware that is required to do the job. And moreover if we use and gate or gate etcetera we know that they are predefined and there is there would not be any problems with that the pre existing designs. So, are they are proven sub blocks. And we have some of this ip course available as libraries or Verilog predefined language. So, these are the advantages of structural modeling.

The disadvantages are that if the design becomes very big and we do not know what hardware a particular functionality will give. So, then it becomes very difficult to use structural modeling. And we may not always know the hardware of the architecture of the design. So, also to get into if at all we decide to structural modeling to know the actual hardware of each functionality is difficult and it will take a lot of time. So, for a higher level design for complex design this is very difficult to work with structural modeling.

(Refer Slide Time: 10:53)

Using primitives

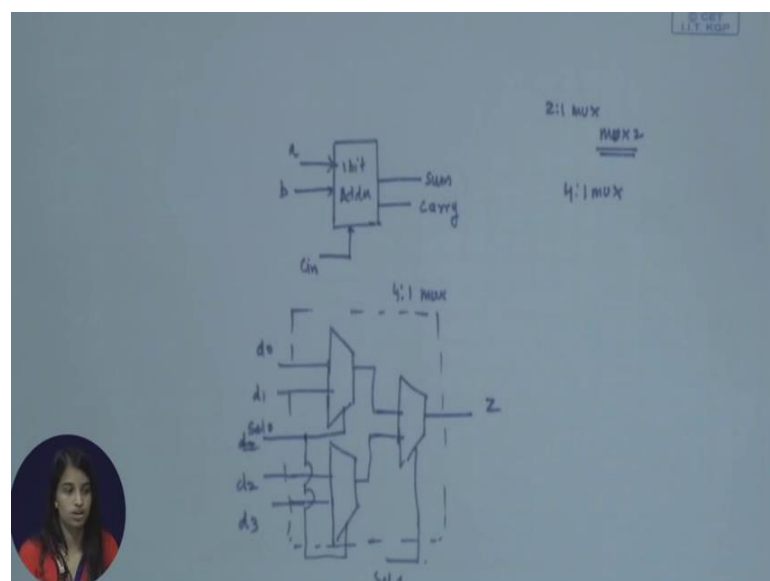
- Pre-defined gates already described in the Verilog language
- Can have only one output but any number of inputs. They need not be declared, they are only called.
- First variable in port list of primitive is always output and naming of primitives is optional.
- Language has no restriction on inputs but depends on technology being targeted to.
- Delays can be specified again optionally.
- Any number of gates of same functionality can be called in single line.
- Basic combinational gates are available as primitives.
- Used directly in the code as smaller case key-words.
- Some examples :and, nand, or, nor, not, xor, xnor.

```
//1-bit Adder
module adder{
  output sum, carry;
  input a, b, cin;
  xor XO (sum,a,b,cin);
  and AO(a0, a,b),
  A1(a1,b,cin),
  A2(a2,cin,a);
  or O(carry,a0,a1,a2);
endmodule
```

CS60087

So, we saw how we use predefined gates earlier in structure modeling, so we call them primitives. So, if you can see here I have taken an example of adder this is a 1 bit adder I have taken which has a sum and carry out put two in three inputs a b and c.

(Refer Slide Time: 11:20)




So, if I draw it I have a 1 bit adder circuit I have a and b as two inputs to be added 1 bit input, c in is a 1 bit carry in and we get sum and a carry out. I will describe how are primitives are used in Verilogs. So, there is a way that the primitives are instantiated or used inside a Verilog model. So, each primitive have inputs and outputs are specified at first; like the first thing that comes during the installation of a primitive is the output. So, we always specify what is output first and then the number of inputs.

Generally primitives have only one output and many number of inputs. So, there are no language restrictions, but generally it depends on the technology that we use. Let the number of inputs and outputs depends on the technology use. So, we can optionally specify delays of the and gates like we can put and hash 1 2 something which defines the propagation delay or other delays in this particular gate. Also the combinational gates these x or and or is readily available in Verilog as primitives.

So, some of the examples of primitives are and, nand, or, nor, not and all these. So, we know now know how to use a primitive in a structural modeling and what are the advantage and disadvantages of structural modeling.

(Refer Slide Time: 13:06)



Component Instantiation

- Instantiation
 - Predefined modules can be called just like primitives
 - An Instance is a unique copy of a pre-defined module.
 - Predefined modules are connected to one another by means of nets.
- Component instantiation can be done in two ways
 - Named port connection / calling by reference.
 - Ordered port connection / calling by position.
- Most simulators /synthesis tools require components to be compulsory named
- Component instantiations do not accommodate delays like primitives.
- While simulating component should have been compiled before it can be called in a top module.

```
// 4-to-1 multiplexer
module mux4(
  input d0,d1,d2,d3,
  input [1:0] sel,
  output z);
  wire z1,z2;
  /*instances must have unique names within
  current module. Connections are made using
  .portname(expression) syntax.*/
  // order doesn't matter...
  mux2 m1(sel[0],.i0(d0),.i1(d1),.z(z1));
  mux2 m2(sel[0],.i0(d2),.i1(d3),.z(z2));
  mux2 m3(sel[1],.i0(z1),.i1(z2),.z(z));
  /*could also write "mux2 m3(z,z1,z2,sel[1])"
  NOT A GOOD IDEA! */
endmodule
```

CS60087

Now, we will see what a component instantiation is. I have been talking about instantiation; instantiation is nothing but using a predefined module in a top module, in another module. Suppose like in the earlier example I spoke about 2 is to 1 MUX so we defined those that 2 is to 1 MUX with the keyword MUX 2. So, we will see how we will use this MUX to create a 4 is to 1 MUX. So, here in the example I am using MUX 2 as a predefined module in the top module MUX 4.


So, if I draw the diagram we have 3 2 is to 1 MUX. So, these are 3 2 is to 1 MUX and we use them to create a 4 is to 1 MUX. So this is a 4 is to 1 MUX, if you see the design we have in a 4 is to 1 MUX we have d 0 d 1 we have d 2 sorry this is not d 2 d 3. And we have an output z we have a select line which is 2 bits. So, we will use one select line for these two MUXes and one select line for this MUX. So, we use 2 is to 1 MUX to create 4 is to 1 MUX. And we will instantiate it 2 is to 1 MUX inside the MUX 4 module and see how component instantiation work.

So, component instantiation can be done in two ways: first thing is it can be called by reference and it can be called by position. It somewhat similar to C function calling, but it is not actually that; I will tell you how it is done. So, the currently whatever you see in this slide in the code this is by reference. So, we provide dot select dot i 0 dot i 1 dot z these are the (Refer Time: 15:41) of MUX 2 we have i 0 i 1 as input we have z as output and one select line. So, we instantiate MUX 2 using calling by reference using this sent this procedure this syntax. So, we specify ports in dot port name and the net that connects it. So, all the MUXes are connected by other wires. So, these are call internal wire nets. So, we have to declare them. So, that is why I have used why wire z 1, z 2 this is a data type that will discuss later.

So, component instantiations like what we spoke about earlier it does not include delay, but it should include a name component name; like I have three MUXes I have named them m 1, m 2, m 3. So, before we do a simulation of this whole model if you want to test this 4 is to 1 MUX if it is operating properly if you want to simulate this you first have to synthesize this individual module and then use them. So, this is how a general component instantiation work. There may be more parameters, more things that can be

added this is the basic things. So, we will see how more Verilog constructs can be added in the later slide.

(Refer Slide Time: 17:14)



Dataflow Modeling

- Dataflow modeling consists of **continuous assignment statements**
 - They start with the keyword **'assign'**. These statements are always active.
 - They are executed whenever there is a change in a variable on the right hand side of the statement.
 - They are used when the system being designed can be completely represented in Boolean equation format.
 - Several assignment statements are concurrent in nature (they execute in parallel).
 - They can also represent the propagation delay from input to output.
- This type of execution model is called **"dataflow"** since evaluations are triggered by data values flowing through the network of **wires and operators**.

```
// 2-to-1 multiplexer with dual-polarity outputs
module mux2(
  input i0,i1,sel,
  output z,zbar);
  // again order doesn't matter (concurrent
  // execution!)
  // syntax is "assign LHS = RHS" where LHS is a
  // wire/bus
  // and RHS is an expression
  assign z = sel ? i1 : i0;
  assign zbar = ~z;
endmodule
```

CS60087

So, now we know what is structural modeling and its disadvantages, now we will discuss about dataflow modeling which is basically modeling the output with respect to input.

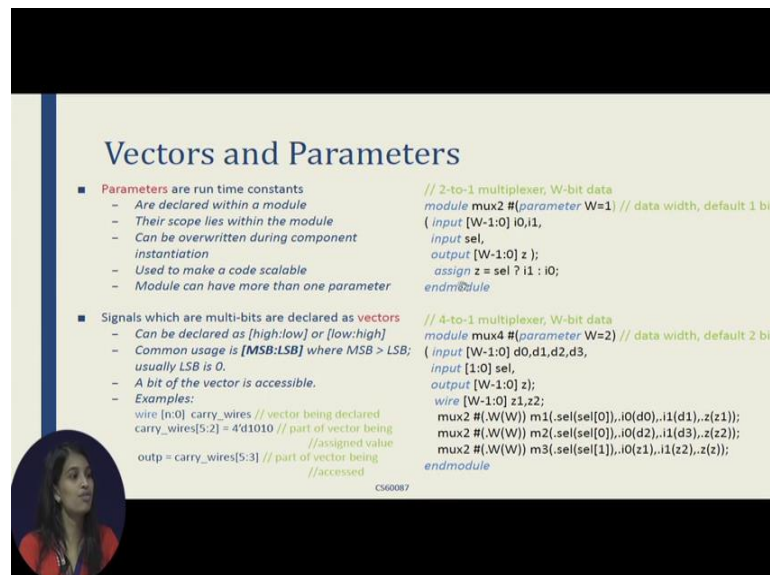
So, all the dataflow modeling code starts with a continuous assignment statement. So, a continuous statement is basically representing output in terms of input. So, we use the **assign** keyword for a continuous assignment statement and they are executed parallelly. If we see the example of MUX 2 instead of using structural modeling I have here used dataflow modeling, I have used the **assign** keyword and I have used operators these are called operators and operations; inputs are called operands and we have operators and we get the output according to the change in the input.

The **assign** statement inside a dataflow modeling is executed parallelly and they can represent propagation delays. So, we can put the **z bar** comes after some five time delay of **z** that we can specify, so I am not putting it here so that you know the basic of dataflow modeling. So, we know now that the execution model called dataflow

has data values flowing through the network through wires and operators. So, basically dataflow modeling is described using a continuous assignment statement which tells us how an output change with respect of input and it uses operators.

So, the main disadvantage of dataflow modeling as it does not have constructs to design the memory element. We can only design a combinational circuit using dataflow modeling it is very difficult to design a memory element using dataflow modeling. So, that is why we came up with the behavioral modeling style and we will see in later slides how we do a behavior modeling model code.

(Refer Slide Time: 19:33)



Vectors and Parameters

- **Parameters** are run time constants
 - Are declared within a module
 - Their scope lies within the module
 - Can be overwritten during component instantiation
 - Used to make a code scalable
 - Module can have more than one parameter
- Signals which are multi-bits are declared as **vectors**
 - Can be declared as [high:low] or [low:high]
 - Common usage is [MSB:LSB] where MSB > LSB; usually LSB is 0.
 - A bit of the vector is accessible.
 - Examples:
 - wire [n:0] carry_wires // vector being declared
 - carry_wires[5:2] = 4'd1010 // part of vector being
 - outp = carry_wires[5:3] // part of vector being

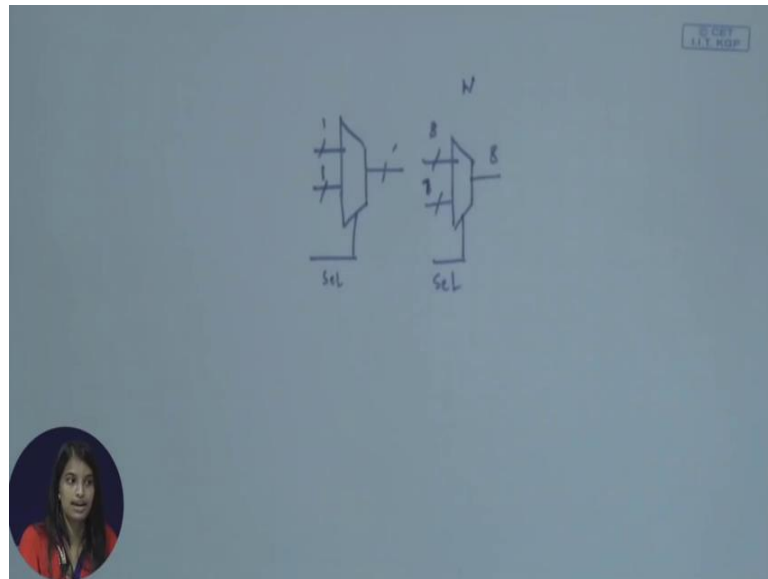
```
// 2-to-1 multiplexer, W-bit data
module mux2 #(parameter W=1) // data width, default 1 bit
( input [W-1:0] i0,i1,
  input sel,
  output [W-1:0] z );
  assign z = sel ? i1 : i0;
endmodule

// 4-to-1 multiplexer, W-bit data
module mux4 #(parameter W=2) // data width, default 2 bit
( input [W-1:0] d0,d1,d2,d3,
  input [1:0] sel,
  output [W-1:0] z);
  wire [W-1:0] z1,z2;
  mux2 #(W(W)) m1(sel(sel[0]),i0(d0),i1(d1),z(z1));
  mux2 #(W(W)) m2(sel(sel[0]),i0(d2),i1(d3),z(z2));
  mux2 #(W(W)) m3(sel(sel[1]),i0(z1),i1(z2),z(z));
endmodule
```

CS600R7

So, before we go into behavioral modeling I want to tell you about more Verilog language constructs and how it has used. So, we will see what are vectors and parameters in Verilog language. So, parameters are nothing but something we want to declare, we want to change in the module after sometime. Suppose we have instantiated the MUX 2 module inside the MUX 4 module and instead of 1 bit input we want some other some 2 bit multiple bit inputs in the MUX.

(Refer Slide Time: 20:12)



For example; we defined this 2 is to 1 MUX. So, in this each input was 1 bit right, but now we have requirement that we want the data to be multiple bit. Suppose this has to be 8 bits and output has to be 8 bits and we have a select line. So, tomorrow this kind of structure for our code to be independent of the change in number of bits we define something call parameter. So, we defined a bit w of the data as shown in the slide. We defined this you using a keyword parameter which is represented like this hash parameter w and by default it takes a value of one for a 2 is to 1 MUX. And when we use it an another module if it is not declared in another module it will take the default value w 1. If you use the w equal to 2 in another module and use this MUX 2 inside MUX 4 then the MUX 2 take the value that has the top module that it takes a value that top module described.

So, it overrides the initial coding that we have done, but it will take the top level parameter value. So, this is how we use a parameter. In this way we can use a parameterized module- say we can define different kinds of MUXes with different inputs you do not have to every time change the basic MUX 2 module to get a multiple bit data MUX. This is how we do modular codes in Verilog so that we can use the single quote for different purposes without changing much of the code. So, hope you have understood how I have used the parameter here. So, this is about the parameter and about the

vectors. So, I told about codes which can be declared here like multiple bits. So, multiple bit signals are called vectors and they can be declared as high to low or low to high like 1 is to 2 or 6 to 1 something like that.

The common usage that we have in practice is MSB to LSB where MSB is greater than LSB and LSB is generally 0, but we can use like we can use part vector in the design. Suppose if the carry wires are declared as n is to 0, suppose n is some 5 some carry wire is 6 bit wire and vector sorry if the carry wire is a 6 bit vector and we have 5 is to 0 as the carry wire declared and we can access each bits separately or we can use a bunch of bits and assign them some more values. So, we can use them to assign it to a output or we can assign them a constant. So, this is how we use a vector in the design.

(Refer Slide Time: 23:35)



Language conventions and semantics

Conventions: <ul style="list-style-type: none">- Verilog is case-sensitive- Some simulators are case-insensitive- Advice: - Don't use case-sensitive feature!- Keywords are always lower case- Different names must be used for different items within the same scope- Identifiers can have<ul style="list-style-type: none">✓ Upper case and lower case alphabets✓ Decimal Digits (but should not start with digits)✓ Underscore (again not at the beginning)	Semantics: <ul style="list-style-type: none">- All statements are terminated with ';'.- Comments :<ul style="list-style-type: none">✓ all characters after '//' in a line are treated as comments✓ multiple line comments begin and end with '/*'- Compiler directives begin with '#'- Built in system task and functions begin with '\$'- Strings enclosed with double quotes must be on one line
--	--

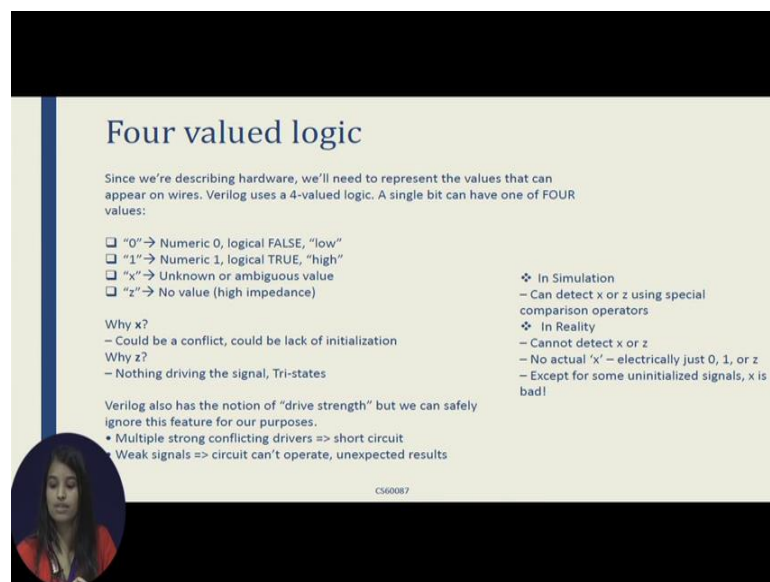
CS60087

Next we will see what are the basic language conventions and semantics. So, Verilog is the case sensitive language and we generally do not use a case sensitive feature because it makes a design very difficult to debug. If in case you have made a mistake it is very difficult to debug. So, all the keywords are in lower case generally. And different name should be used for different designs, like I have used MUX 2 for 2 is to 1 MUX, MUX 4 for 4 is to 1 MUX; so we should use different names of different scope. And the

identifiers like the names that I used for a module or for instantiated module should have uppercase or can have uppercase and lowercase alphabets, decimal digits or underscores.

And in the semantics that we can use in a Verilog language as that every line of the code is ended with a semicolon. We can have two types of comments; we have compiler directives which I will come to later. We have built in system tasks and functions which start with dollar. And we have strings and codes in double colon.

(Refer Slide Time: 24:51)



Four valued logic

Since we're describing hardware, we'll need to represent the values that can appear on wires. Verilog uses a 4-valued logic. A single bit can have one of FOUR values:

- ❑ "0" → Numeric 0, logical FALSE, "low"
- ❑ "1" → Numeric 1, logical TRUE, "high"
- ❑ "x" → Unknown or ambiguous value
- ❑ "z" → No value (high impedance)

Why x?
– Could be a conflict, could be lack of initialization

Why z?
– Nothing driving the signal, Tri-states

Verilog also has the notion of "drive strength" but we can safely ignore this feature for our purposes.

- Multiple strong conflicting drivers => short circuit
- Weak signals => circuit can't operate, unexpected results

❖ In Simulation
– Can detect x or z using special comparison operators

❖ In Reality
– Cannot detect x or z
– No actual 'x' – electrically just 0, 1, or z
– Except for some uninitialized signals, x is bad!

CS60087

So, in Verilog we spoke about wires. So, wires are generally four valued logic and they can take 4 values, we are designing hardware. So, hardware will particular hardware will have nets or wires connected between each other. So, we have to think about what the values in the hardware it can take. So, generally we know about 0 and 1 which corresponds to a voltage that are plus 5 or minus 5 or plus 3.3 or minus 3.3, but we do not know about x and z. But the Verilog language allows us to define a signal as with following values 0 1 x and z 0 belonging to low 1 belonging to high x belonging to an unknown value and z belonging to a high impedance state of the signal.

So, x is generally used in simulation in practice or in hardware that is nothing called as x. And z means high impedance that the line is not driven by anything or it is idle. So, these

are the values that we use in Verilog code for a particular wire or net. So, why we use x is that we can have multiple signal driving a same signal. So, to find the conflict between that we use x and we initialized sometimes values as x so that we can distinguish between the later values that it can take. And this is generally used in simulation. Then why we do you use z? We use z so that we know that no other signal is driving it and the line is idle.

Verilog also has some other drive strength properties that we can use with the signals which will not discussing which is not generally used. So, we now know that Verilog has signals which are represented in four valued logic. I think we are out of time now. So, I will stop this session here and I continue this in the next session.