Hardware Modeling using Verilog

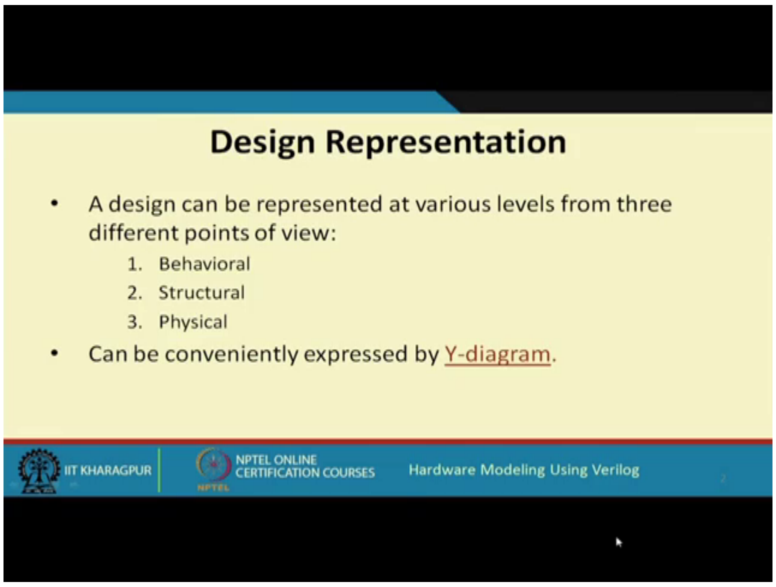
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 02
Design Representation

In this lecture we shall be talking about Design Representation. Now, earlier I had said that you can create a design using a hardware description language; verilog with respect to this course. Suppose we create a design in some hardware description language like verilog; now the question arises at what level do it typically create the design; or how do we create? How do we visualize a design?

So, when we talk about design representation; what you mean is that, given a design; how we can visualize the same design from different angles? Now, the way we shall be presenting here is that there are three distinct ways or three distinct angles from where you can visualize the same design and the implications will be a little different; let us see.

(Refer Slide Time: 01:22)



Design Representation

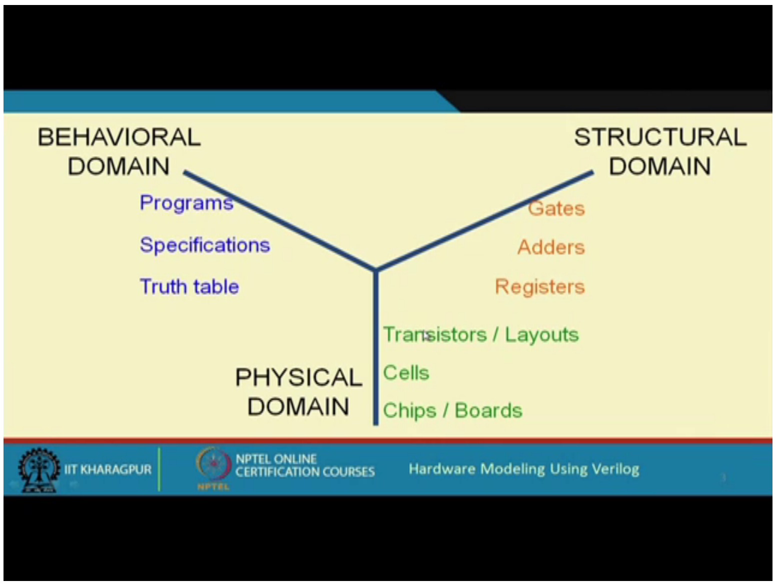
- A design can be represented at various levels from three different points of view:
  1. Behavioral
  2. Structural
  3. Physical
- Can be conveniently expressed by Y-diagram.

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

So, design representation as it said this talks about how we represent a design. A design is nothing but you can think it as a black box that is trying to perform its intended operation. You have designed that block to carry out certain operation and that is your design. Now, design representation we are considering at three different levels; one you are calling it as behavioural, structural and physical.

Now, with respect to verilog coding these terms behavioural and structural will be coming repeatedly and we will be understanding the exact differences and how we can code behavioural and structural designs in verilog in a very efficient way and of course, the third representation is physical. Now, there is a very interesting way to just express this design views or viewpoints, it is with respect to something called a Y-diagram.

(Refer Slide Time: 02:39)



Now, this name has come because it has a shape of a Y. So, you think of an imaginary Y with the three design representations; in the three directions. Let us say; this is our behavioural domain, this is our structural domain and this is our physical domain. Now, just along each of these domains; like for example, in the axis of behavioural domain. Here we are talking about a design in terms of its behaviour; like in a very high level case, you can think of an algorithm or a program; you can think of a specification, you can think of Boolean expressions, you can think of truth tables; state diagram and so, on; these are all examples of behaviour.

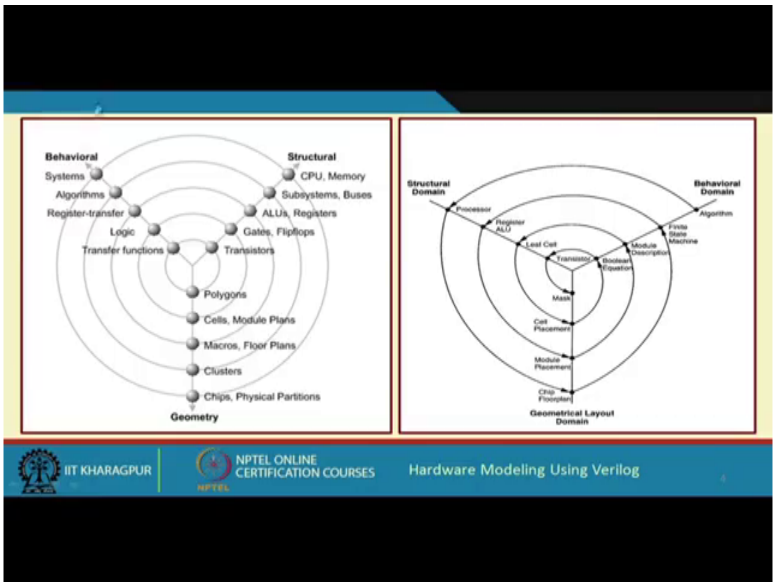
Now, when we talk about structural domain; now well we will we had explained in the last lecture what is a net list and net list is nothing but some building blocks and their interconnections. Now, this building blocks can defined at various different levels; depending on that our structural domain specifications can vary. So, it can be at the register transfer level, it can be at the functional level, it can be at the gate level, transistor level and so, on. And talking about the physical domain; here you are actually talking about the implementation. Like in the ultimate case, when the final design is complete; we would be getting our chips or a printer circuit board.

Before that our designs will be placed on a silicon layout in terms of cells. Now, what will those cells contain? They will contain some transistors; which will have some layout. So, these are all implementation or physical design related things; that our chip the cells that our put on a chip and the transistors that make a cell. So, here we are looking from a angle which talks about the actual implementation.

So, in one side was the behaviour; one side was the net list view and the other side was the actual physical implementation view. So, these are the three viewpoints; now there is some interesting ways to look at these Y; let us look at it once more from this point of view.

Here we have showed it in a slightly different way; behaviour, structural, geometry and you see we have drawn some concentric circles. The outer most circle indicates the most abstract design, so at the behavioural level; this outside circle indicates systems is like in highest level my system a consist of two processors one memory and one I O module; this is my highest level design of the system.

(Refer Slide Time: 05:55)



So, this is my behaviour; so, this in the structural level will be a net list that consisting of interconnection of CPU’s and memories as I have said the example and in the physical level; this will consist of some chips a collection of chips one may be processor, one may be memory and so, on. So, as you go to the inner circles you are carrying out synthesis, you are refining your design like at the next level your behaviour is your algorithms. Here you have subsystems and buses, here you have some clusters; next level you have register transfer level blocks; you have ALU’s and registers net list level and here you have macros and floor plans.

Next level inside you have logic cells; gates and flip flops. So, here you have connection of gates and flip flops, here you have cells module plans and at the lowest level you have the transfer functions, transistors and polygons. So, as you go from the outer most circle towards the inner most circle; from all the three viewpoints your design is getting refined, from very high level or abstract see from a very abstract level in the behaviour; you will say the well I want a computer system.

In the structural level, you will say that well I need two processors and one memory more block. Now in the physical level you will say that well on the board I need four chips; these are the different ways to look at the same thing. So, as we have move towards the centre of the circle each of these blocks will get further refined, they will get more detailed out and will be having some kind of a top down design process.

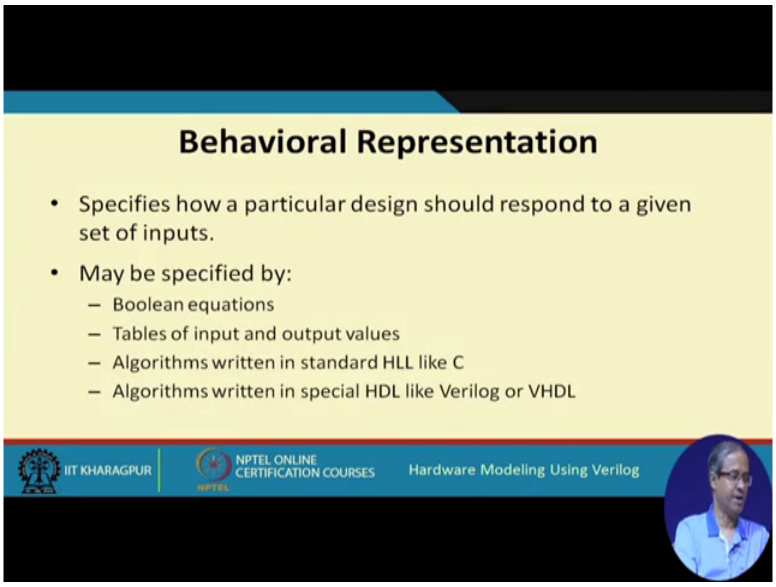
Now, if you think of it well we normally do not do it this way that at one level; we have all the three viewpoints; then we go to the next level, again look at all the three viewpoints. So, we really do not work exactly in that way; rather we work in an alternate way which is shown in the diagram on the right. So, here I am showing the behaviour axis here; structural here and geometric here or the physical here. Now you see; here I am giving an example, we start with an algorithm at the behavioural level; so, highest level is algorithm.

So, we carry out some transformations in a systematic way which is represented you see as a helix here; from the outside you slowly move towards the inner most point of this of this structure. So, from the algorithm; you translate into processer which you translate into a chip floor plan and each of the blocks in a chip floor plan you translate into finite set machines; they will be translated into registers and ALU's, they will be translated the physical level in to modules which will be placed module placement.

Then each of these modules; will be having a description at the behavioural level module description now each of the modules will be designed as a net list we call them as a leap cell they will be placed cell placement. Now, each of the cell will be expressed as Boolean expressions they will be realize is in transistors and finally, there will be realized using the layout masks.

So, you see this is this helical structure is more natural in terms of the steps that a designer actually follows. So, although that concentric circle is a good way of representing, but the actual process that you follow is this helix; from the outermost point we slowly move towards the centre of the Y; this is the so, called top down design process.

(Refer Slide Time: 10:07)



Behavioral Representation

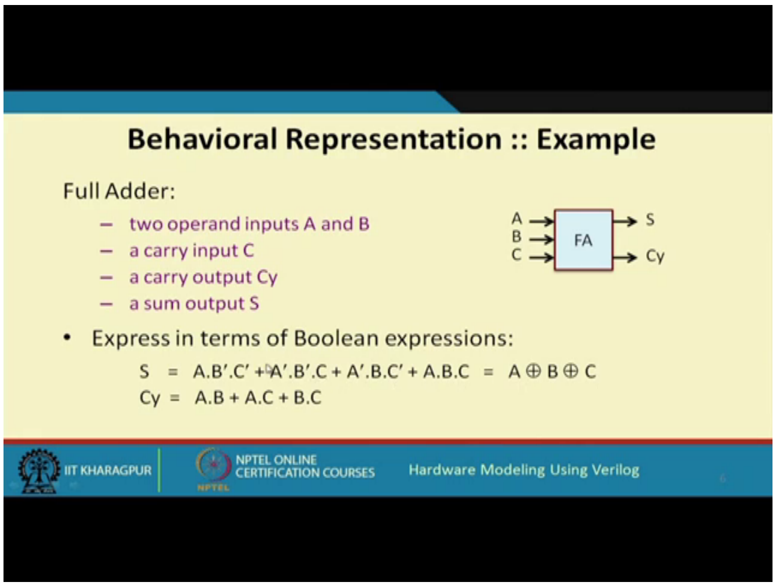
- Specifies how a particular design should respond to a given set of inputs.
- May be specified by:
  - Boolean equations
  - Tables of input and output values
  - Algorithms written in standard HLL like C
  - Algorithms written in special HDL like Verilog or VHDL

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, let us again come back to the behavioural representation; here we shall be showing you some examples in verilog. So, just to repeat in the behavioural representation we specify; how a particular design should work; that means, given a set of inputs; what should be its output. So, here we are not telling that exactly how the design has to be implemented. Some examples of behavioural representation are Boolean expressions, truth tables table of input and output values.

So, you can write algorithms in some standard high level language like C that is also behaviour or algorithms written in hardware description languages like verilog or VHDL these are all examples of behavioural representation.

(Refer Slide Time: 11:01)



Behavioral Representation :: Example

Full Adder:

- two operand inputs A and B
- a carry input C
- a carry output Cy
- a sum output S

- Express in terms of Boolean expressions:

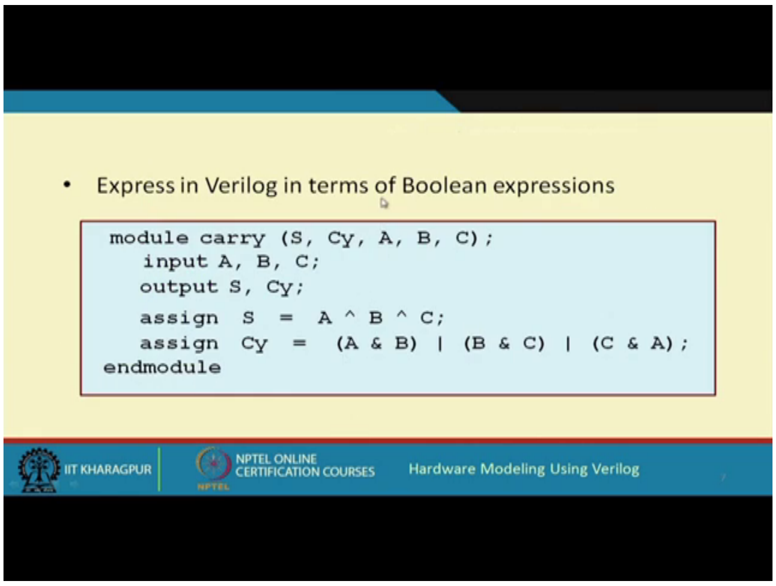
$$S = A.B'.C' + A'.B'.C + A'.B.C' + A.B.C = A \oplus B \oplus C$$
$$Cy = A.B + A.C + B.C$$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

Now, let us take an example a simple example of a full adder. So, a full adder we will have two inputs A and B and a carry input C. So, it will be generating a carry output and a sum like this; A, B, C are the inputs and S and C y are the outputs. Well, now I am not trying to explain how a full adder works; you already know the basic definition of full adder; I am just writing the logic expressions of the sum and carry. So, the sum and carry expression in terms of A, B, C will be this; A, B bar C bar or A bar, B bar, C or A bar, B C bar or A, B, C; which is nothing but A exclusive or B exclusive or C.

So, it is the exclusive or of the three inputs that is sum and carry is nothing but A, B or a C or B, C. So, this is one way of expressing the behaviour of a full adder; because here you are not specifying what kind of gates to use; how many gates to be used and so, on; we are just writing now on the expressions and we are telling that well this is your sum and this is your carry; this will be the behavior, this is an example.

(Refer Slide Time: 12:33)



- Express in Verilog in terms of Boolean expressions

```
module carry (S, Cy, A, B, C);
   input A, B, C;
   output S, Cy;
   assign  S  =  A ^ B ^ C;
   assign  Cy  =  (A & B) | (B & C) | (C & A);
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

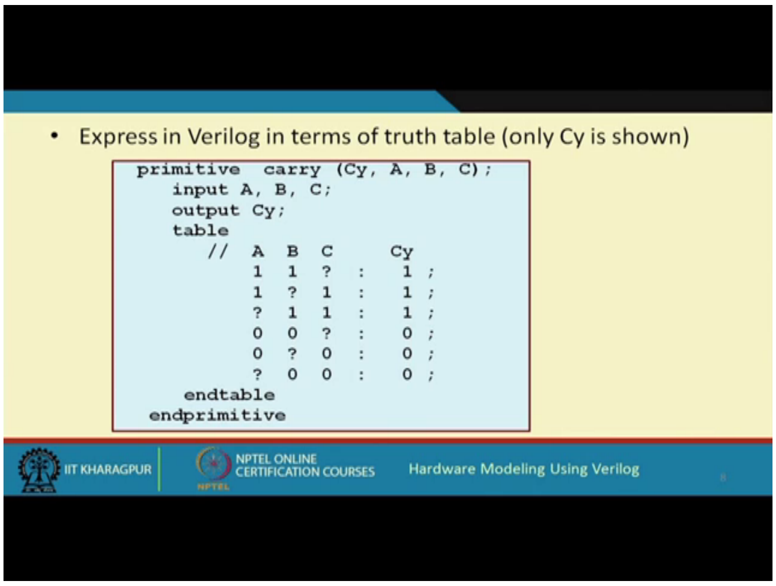
Now, this is in terms of Boolean expression the same thing you can express in verilog as follows well I have not talked about the syntax of verilog.

So, possibly this is your first look at a verilog program; let us see what it contains. The syntax is some more similar to the language C; here every statement ends with a semicolon. This is like a function in C; this is called the module. So, verilog program will consist of one or more modulus. So, it starts with this keyword module; it ends with end module this is this module is followed by the name of the module and the parameters. The parameters are well in terms of the hardware these are nothing but the input and the output ports of that block.

Now for a full adder; the inputs are A, B, C and the outputs are carry out and sum. So, this S; C y, A, B, C are the five parameters; we declare; A, B, C as input like this, input A, B, C; output S; C y. So, that the cad tool will know that which are your input pins input signals which are your output signals and there is a assign statement available in verilog; here we are showing it using assign you can directly write down the Boolean expression.

This hat is the expression for exclusive or this indicates xor. So, sum is A, xor B, xor C and this amp person is end bar is or carries a B or B C or C A. So, you see in verilog you can express Boolean expression; which is behavioural specification very concisely in this way using the assign statement.

(Refer Slide Time: 14:42)



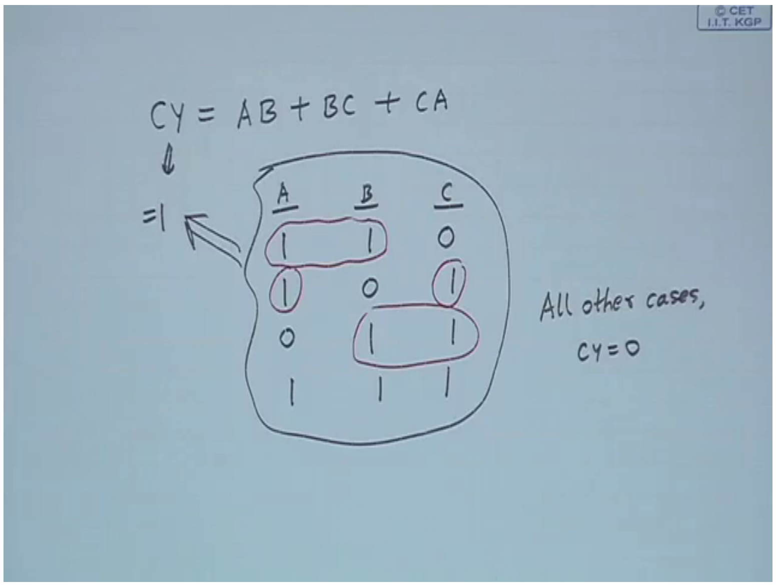
- Express in Verilog in terms of truth table (only Cy is shown)

```
primitive  carry (Cy, A, B, C);
   input A, B, C;
   output Cy;
   table
      //  A  B  C     Cy
          1  1  ?  :  1 ;
          1  ?  1  :  1 ;
          ?  1  1  :  1 ;
          0  0  ?  :  0 ;
          0  ?  0  :  0 ;
          ?  0  0  :  0 ;
    endtable
 endprimitive
```

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

So, this is one way to specify a behavioural specification in verilog. Now, this same design; the full adder I am showing only one of the output the carry, well just if we recall the carry is nothing but A B or B C or C A.

(Refer Slide Time: 14:55)



So, you tell me when the carry will be 1? The carry will be 1 for several conditions of A, B and C; when either A and B is 0, but C is; A B is 1, C is 0; A and C is 1; B is 0 or B and C is 1; A is 0 or all these three are 1.

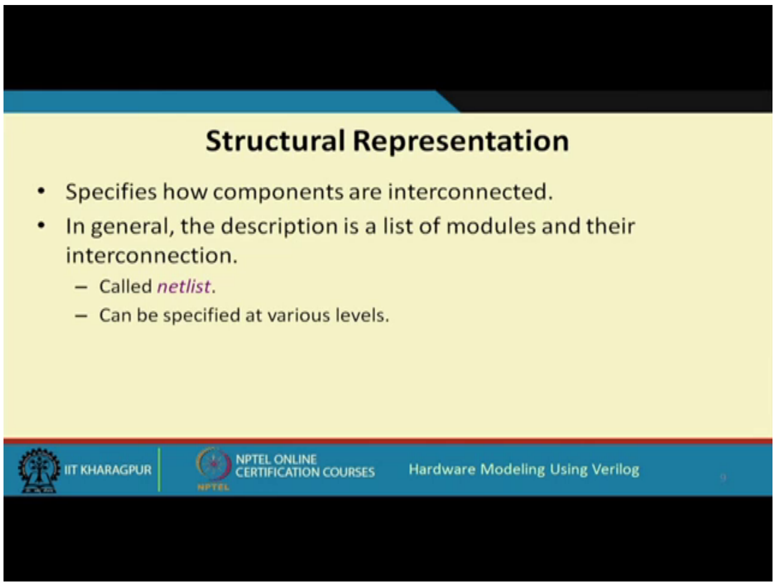
So, for these four cases my carry will be 1; for all other cases carry is 0. So, this I can express or as a as a truth table. So, how do I express? Well I need not have to specify all the rows; I can simply say that at least two must be 1; A B, A C or B C; at least two. If it is 1; then the carry will be 1; this is what you are specify here you see; in this truth table.

So, this is again way to express a behavioural specification in verilog in terms of the truth table. So, there is a keyword called primitive. So, we use this primitive keyword here; so, I am only showing for carry. So, there are four parameters this is the output A, B, C are the inputs. So, I declare the input and output signals. So, I use a keyword table; so, here this is a comment just for showing that these are the values of A, B, C and this is C y this is a comment line and just for convenience.

And the way you specify the input is you specify 0 or 1 and question mark means don't care and colon separates inputs and outputs. The first line says; if A and B there 1, but C is don't care; then carry is 1; like you see here I means if A and B are 1, C is 0 then also carry will be 1, but if A and B are 1, C is 1; then also carries 1. So, actually C is a don't care; it does not depend on C. So, this we can write in a concise way using the don't care notation.

Similarly, if A and C are 1; B is don't care or B and C are 1; A is don't care; then also carry will be 1. But on the other cases at least two of the inputs are 0; A B 0 or A C 0 or B C 0; other is don't care; then carry will be 0. So, you see; if we use don't care then instead of the eight rows of the truth table; here we require only six some of specification may be shorter.

(Refer Slide Time: 18:08)



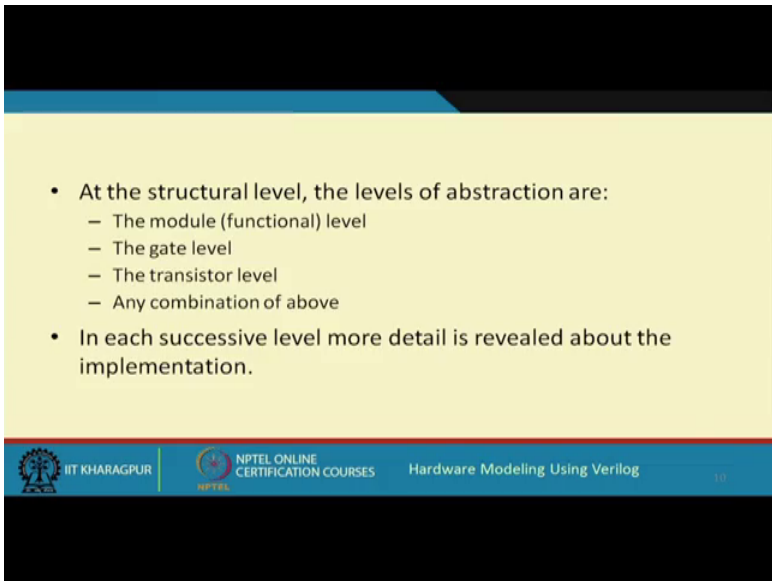
Structural Representation

- Specifies how components are interconnected.
- In general, the description is a list of modules and their interconnection.
  - Called *netlist*.
  - Can be specified at various levels.

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

Now, let us move on to structural specification; now let us structural specification as I said that; it actually specifies how some modules are interconnected. So, when we say that I am defining something in a structural way; which means I will have to specify the modules and I will also have to tell how their actual interconnected. Let us see an example on this. So, earlier we mentioned that; this kind of structural representation is also called the net list. And net list can be specified at various levels at a every high functional level, gate level, transistor level and so, on.

(Refer Slide Time: 18:58)



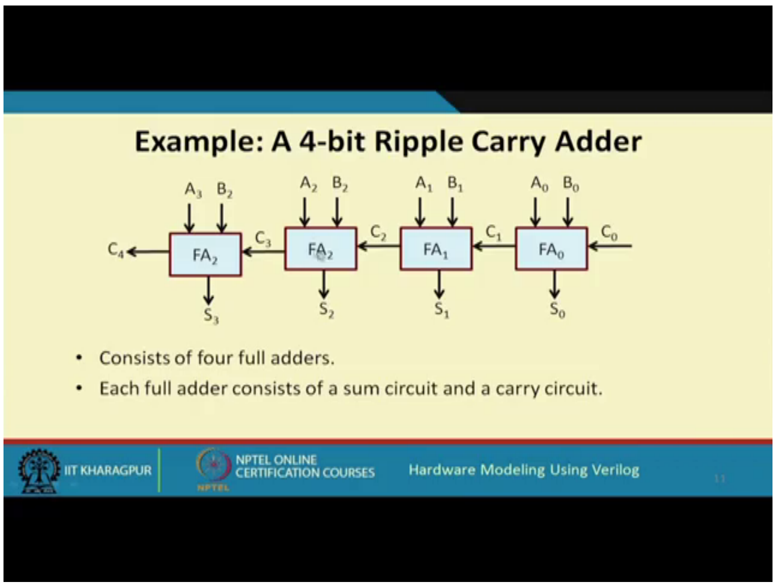
- At the structural level, the levels of abstraction are:
  - The module (functional) level
  - The gate level
  - The transistor level
  - Any combination of above
- In each successive level more detail is revealed about the implementation.

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

This will see later slowly; so, structural level; levels of abstraction is also we talked about earlier I mention gate level, transistor level or can be hybrid design also.

Some of the blocks can be the gate levels, some of the blocks can be at higher level. So, as you move down; more and more details I revealed. Like net list; which is at a functional level; multiplexers, decoders, adders their number of blocks will be less. So, as you move down to the gate level list; net list equivalent net list, number of gates will be much larger. As you move down again to a transistor level net list; number of transistor shall be even larger. So, as you move down; your size of the net list will be increasing gradually. Let us take an example to illustrate structural design.

(Refer Slide Time: 19:53)



Example: A 4-bit Ripple Carry Adder

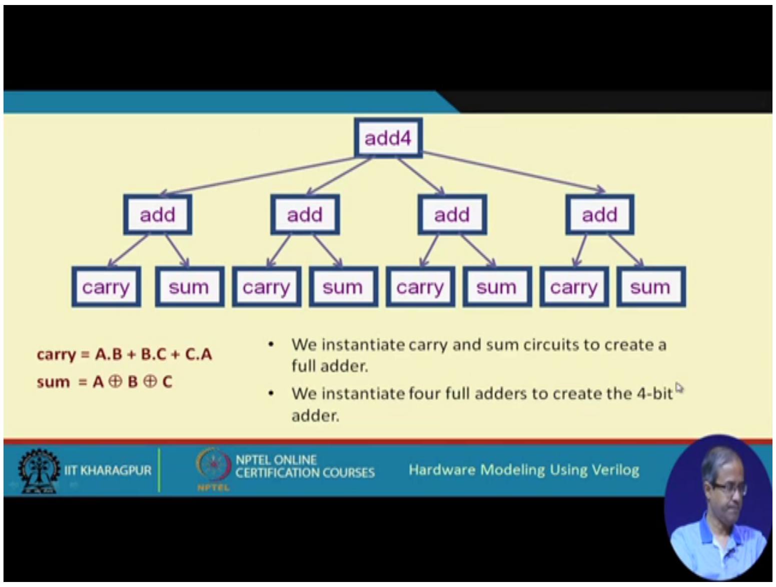
- Consists of four full adders.
- Each full adder consists of a sum circuit and a carry circuit.

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

Now, this is a circuit which you must be familiar with; this is a 4 bit ripple carry adder. So, what we have done? Just to recall. So, 4 bit ripple carry adder consists of 4 full adders. So, I want to add two numbers A 0, A 1, A 2, A 3 and B 0, B 1, B 2, B 3; they are all 4 bit numbers. So, I feed the corresponding bits to the 4 full adders and C 0 is my carry in. The carry out from the first full adder will be going as carry into the next carry out from here you will go to the carrying of the next; similarly here and C 4 will be the final carry out and S 0, S 1, S 2, S 3; will be the final sum.

So, if you look inside the full adders; where just now sometime back we saw, the behavioural specification of a full adder; we saw that there is a some part there is a carry part, there will be one circuit which will be computing the sum; there can be another circuit disjoint or sheared may be which will compute the carry. So, each full adder we will consist of a sum circuit and a carry circuit.

(Refer Slide Time: 21:13)


add4
add
add
add
add
carry
sum
carry
sum
carry
sum
carry
sum
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

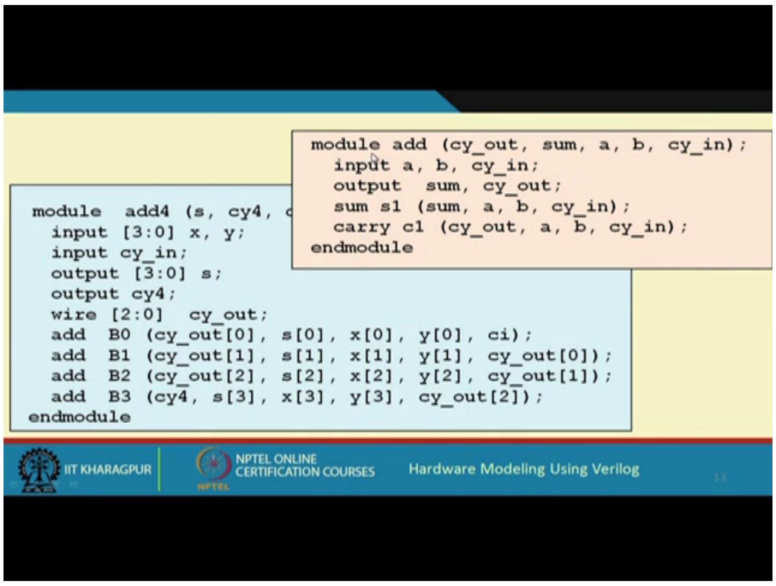
carry = A.B + B.C + C.A

sum = A ⊕ B ⊕ C

- We instantiate carry and sum circuits to create a full adder.
- We instantiate four full adders to create the 4-bit adder.

So, conceptually speaking we show it like this. So, as if we have a 4 bit adder; we call it as a 4 bit adder. 4 bit adder consists of 4 full adders add this is a full adders; each full adder consists of a carry circuit and a sum circuit; carry circuit, sum circuit. So, this is how we can describe a circuit in a hierarchical way and also structural way. Like every adder will be a combination of carry and sum and this 4 bit adder will be a combination of 4 full adders. Now, this carry on sum we already know these are the expressions. So, let us see in verilog how we can do this.

(Refer Slide Time: 22:03)



```
module add (cy_out, sum, a, b, cy_in);
  input a, b, cy_in;
  output  sum, cy_out;
  sum s1 (sum, a, b, cy_in);
  carry c1 (cy_out, a, b, cy_in);
endmodule
```

```
module  add4 (s, cy4, c
  input [3:0] x, y;
  input cy_in;
  output [3:0] s;
  output cy4;
  wire [2:0]  cy_out;
  add  B0 (cy_out[0], s[0], x[0], y[0], ci);
  add  B1 (cy_out[1], s[1], x[1], y[1], cy_out[0]);
  add  B2 (cy_out[2], s[2], x[2], y[2], cy_out[1]);
  add  B3 (cy4, s[3], x[3], y[3], cy_out[2]);
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

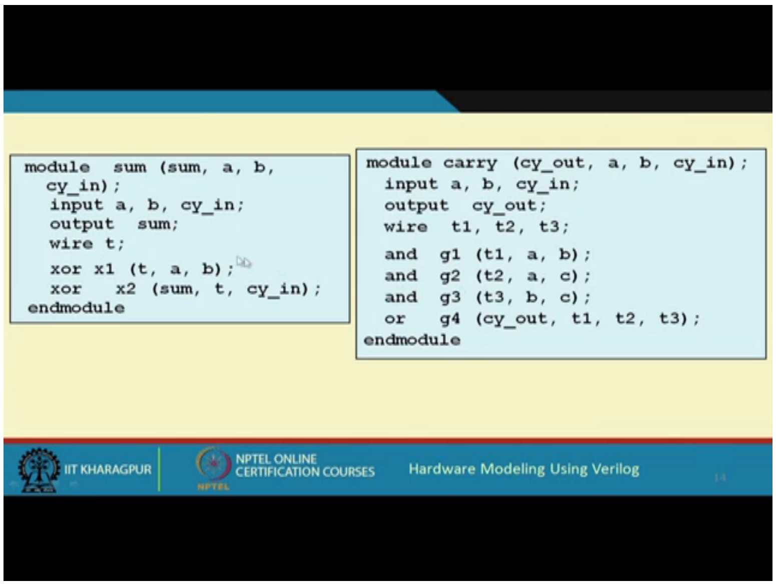
So, here I am showing the top level module of a 4 bit carry look at adder. So, here the parameters has specified here; well here I am not going into the detail because all these details we shall be explaining again later. I am just showing you; how this verilog code looks like. Just one thing you see here; here I have declared x and y are the inputs and this x and y input; I have declared as a vector 3 colon 0 means; it is a four bit number, but the bits are number from the most significant side, 3, 2, 1, 0; this is the way to specify in verilog 3 colon 0.

Carrying in a single carry in; similarly output sum is also 4 bit; 3 to 0. See; we here let us show here I mean say we have an adder, we have also defined a full adder; add is a full adder. Full adder has parameters carry out sum and the three inputs. So, you see here in the 4 bit ripple carry adder; I have made four copies of this full adder, this is called instantiation.

Well in a C program; when you call a function. So, the control goes to the function; the function gets executed and again we come back. But in terms of hardware; if I call a full adder four times it means I am using four copies of the full adder, this is called instantiation. So, here four copies of the full adder will be instantiated and I am giving different names B 0, B 1, B 2, B 3 to the four copies and these parameters the way I have given the name; this defines the interconnection.

Like here for example, for B 1 the carry out C I out 0 this will be C I out 0. So, this will be connected to this say about 2; C out 1 will means for B 0, the carry out will be connected to the carry in of the next, for B 1 ; the carry out will be connected to the carry in of the next for B 2 the carry out will be connected to a carry in of the next and so, on because this is your carry in the last parameter. And here this is a high level design the full adder also we have defined in a structural way, you have put a sum and a carry. Sum is module; carries the model we have instantiated them.

(Refer Slide Time: 24:58)



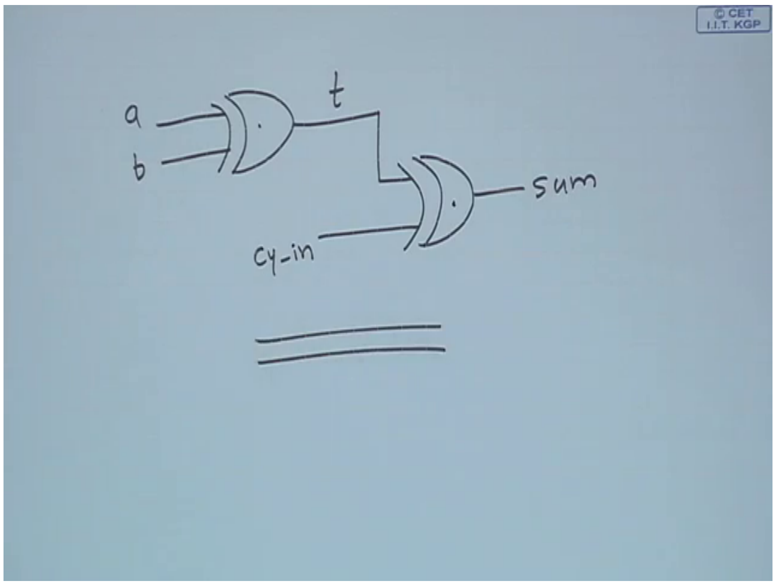
```
module  sum (sum, a, b,
  cy_in);
   input a, b, cy_in;
   output  sum;
   wire t;

   xor x1 (t, a, b);
   xor   x2 (sum, t, cy_in);
endmodule
```

```
module carry (cy_out, a, b, cy_in);
  input a, b, cy_in;
  output  cy_out;
  wire  t1, t2, t3;

  and  g1 (t1, a, b);
  and  g2 (t2, a, c);
  and  g3 (t3, b, c);
  or   g4 (cy_out, t1, t2, t3);
endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

And sum and carry similarly, can be described like this. So, I have defined this in a structural way; not in a behavioural way you see, sum this A, B, C in the inputs; sum is the output and t is a temporary wire.

(Refer Slide Time: 25:27)

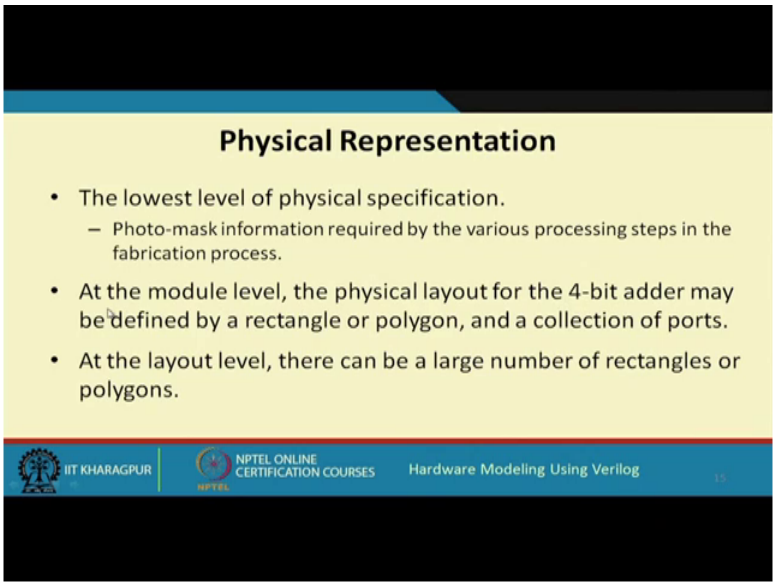


This xor is a gate; what does it t a b means? t a b means the first one is the output second two are the inputs; which means that I have an xor gate for the output is t and the inputs are A and B, the first line indicates this. Then next line indicates; there is another xor sum sum is the output inputs as t and C y in. So, t is there this is another xor gate, this is another input C y in and you generate sum. So, you actually specify this circuit; this is structural because you have specified two gates xor gates and how they are interconnected right. So, this is a structural description of a sum.

Now, these gates xor or for carry this and or these gates are already; there as a part of the verilog language. So, this we shall be studying slowly. So, I have just shown you example here just to show you how a verilog module looks like. So, in the carry block in a similar way; we need four gates g 1, g 2, g 3; this will do a end of AB, end of A C and end of B C and the outputs are t 1, t 2, t 3 and finally, you do a or of t 1; t 2; t 3.

And now you see the number of inputs to the gates you can vary like in the last case the first one will be the output and the remaining three will be the inputs. So, this will be automatically taken like this.

(Refer Slide Time: 27:03)



Physical Representation

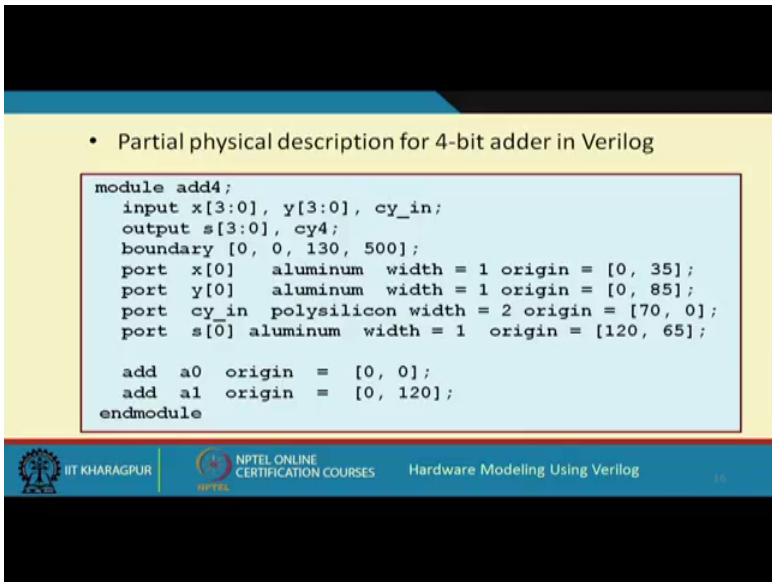
- The lowest level of physical specification.
  - Photo-mask information required by the various processing steps in the fabrication process.
- At the module level, the physical layout for the 4-bit adder may be defined by a rectangle or polygon, and a collection of ports.
- At the layout level, there can be a large number of rectangles or polygons.

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

Now talking about the physical representation which is the lowest level; here as it said that when you finally, fabricate the chips; you define a large number of various polygonal shapes; for the different layers of fabrication.

So, these are this specification of so called photo masks that are required in the fabrication process. Now this 4 bit adder is the way we look at it; from the function or the structural point of view, but from the physical point of view there will be a large number of rectangles or polygons.

(Refer Slide Time: 27:48)



- Partial physical description for 4-bit adder in Verilog

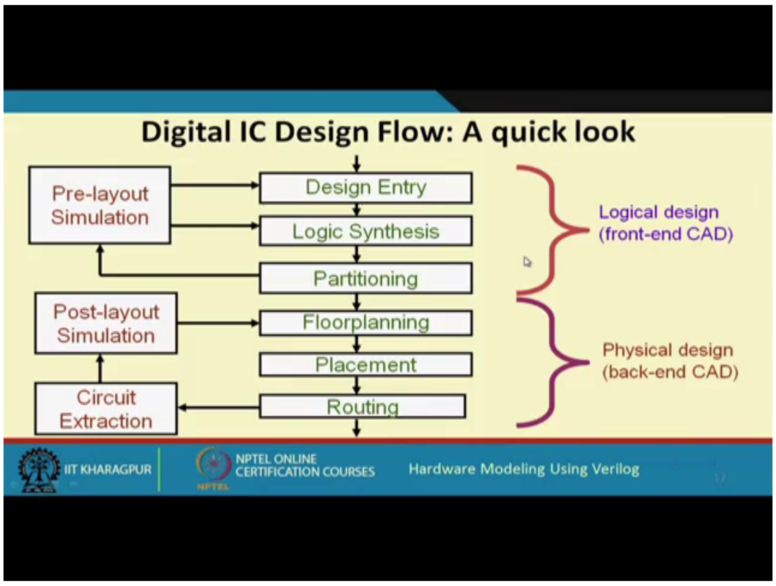
```
module add4;
   input x[3:0], y[3:0], cy_in;
   output s[3:0], cy4;
   boundary [0, 0, 130, 500];
   port  x[0]   aluminum  width = 1 origin = [0, 35];
   port  y[0]   aluminum  width = 1 origin = [0, 85];
   port  cy_in  polysilicon width = 2 origin = [70, 0];
   port  s[0] aluminum  width = 1  origin = [120, 65];

   add  a0  origin  =  [0, 0];
   add  a1  origin  =  [0, 120];
endmodule
```

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

So, this also can be specified in verilog; I am just showing you an example this is not the complete description, just a partial description for the adder. Like you can actually define some coordinates 0, 35 is a coordinate; this is an aluminum wire of width one unit, this is the poly silicon wire of width 2 units and so, on. So, in this way you can define various wires of different metals or materials you can specify the widths basically there rectangular shapes.

Rectangles there will be a large number of such rectangles we will define you will specification. So, I am just showing you these primitives are also means available in verilog. So, when you go down to the layout level; the same verilog language can be used to specify your layout. This is what I meant by saying is that you have a hardware description language and as the process of synthesis continues you transform; one version of verilog to another version of verilog, that another version of verilog to another define version of verilog in that way you proceed and at the final step you get a description which is your final layout description.

(Refer Slide Time: 29:09)


Digital IC Design Flow: A quick look
Pre-layout Simulation
Design Entry
Logic Synthesis
Partitioning
Post-layout Simulation
Floorplanning
Placement
Circuit Extraction
Routing
Logical design (front-end CAD)
Physical design (back-end CAD)
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

So, just a quick look at the digital IC design flow once more; starting from the specification you design entry or this may be your specification in your verilog or VHDL; logic synthesis and there are some steps were you are actually going for the physical design; this is called physical design. You go for partitioning, floor planning, placement routing steps like this.

And in the first steps; you go for design entry, logic synthesis, partitioning. This steps are typically called front end design or logical design and these are called physical design or back end design. And there is some feedback connection also because at some step; you may find that it is not working properly do simulation, you may have to go back and make some changes.

Like here also you can do some circuit extraction and simulation; you say that your delay is not coming proper you may have to go back and make some changes. So, just a very rough view of the whole process of design; front end design and back end design; this entire process have to be traversed by a designer to actually design a circuit in terms of the final layout description, starting from the behaviour.

So, with this we come to the end of the second lecture. Well, in this lecture we have basically talked about design representations; the three ways you can visualize a design and we have seen some examples in verilog; how verilog can be used to capture the design at the different levels of abstraction; different views, behavioural structural and physical.

Thank you.