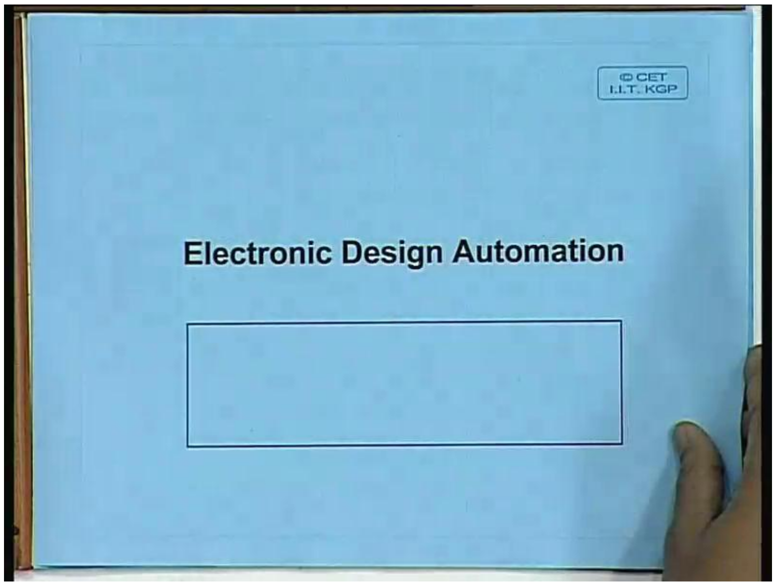
Electronic Design Automation

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

Lecture No #1
Introduction

So electronic design automation, this is the subject we would be talking about in the course of the next series of lectures.

(Refer Slide Time: 01:13)

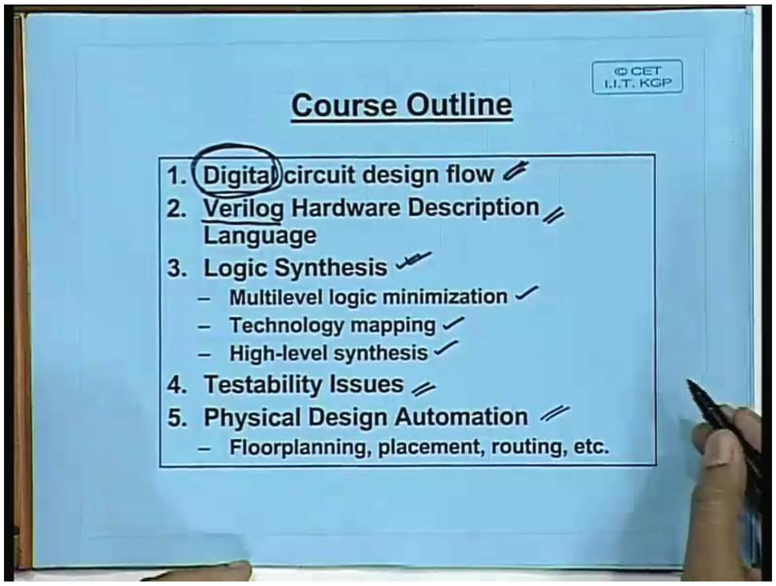


© CET
I.I.T. KGP

Electronic Design Automation

Before I go into the actual series of lectures let me first tell you what exactly we intent to cover in this series of lectures. Well electronic design automation as the name implies what it really means is that we have some kind of a design which we want to carry out starting from the specification we want to arrive at the final design implementation. It can be an FPGA. It can be ASIC. It can be anything.

(Refer Slide Time: 01:40)



©CET
I.I.T. KGP

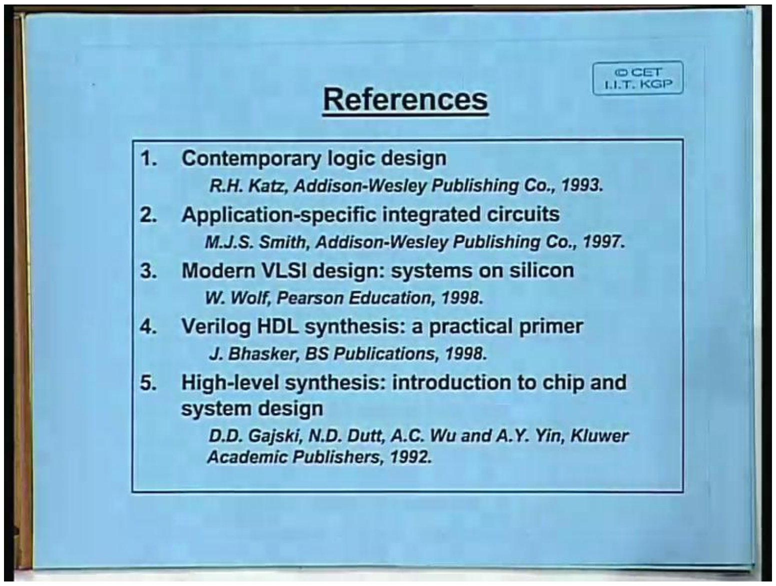
Course Outline

1. Digital circuit design flow
2. Verilog Hardware Description Language
3. Logic Synthesis
   - Multilevel logic minimization
   - Technology mapping
   - High-level synthesis
4. Testability Issues
5. Physical Design Automation
   - Floorplanning, placement, routing, etc.

So let us first try to have a quick look at what are the topics that we intend to cover in this course. First of course we would be looking at the digital circuit design flow which means that if we have a digital design digital circuit specification. What are the different steps that we need to follow in order to arrive at the implementation. Well out here we would be considering only digital circuits. We would not be considering analog designs. Next we would be looking at popular hardware description language namely Verilog. This description language we would be using as our vehicle in some of the future lectures. All examples would be given in Verilog. Next we would be talking about some topics on logic synthesis. Now assuming that you are already familiar with 2 level logic minimization we would be starting from that point onwards. We would be talking about multilevel optimization.

A process called technology mapping and of course something called high level synthesis. Well in high level synthesis we are trying to arrive at a design a register transfer level design starting from a behavioral specification. Then of course we would be considering or talking about a very important issue that once we have designed the circuit up to a particular level how do we test it. So some issues related to testability and at the very end we would be talking about physical design automation which concerns the issues that comes in at the last phase of the implementation namely here we talk about the floor planning placement routing. These issues are arising well. Well when we consider the silicon floor, the silicon floor how do you place the different components on the floor of the silicon. How do you interconnect them? These issues would come in here. Okay. So let me give you a list of some references. Of course this is course so diverse. It is not possible to specify or a few books to cover all of them.

(Refer Slide Time: 04:04)



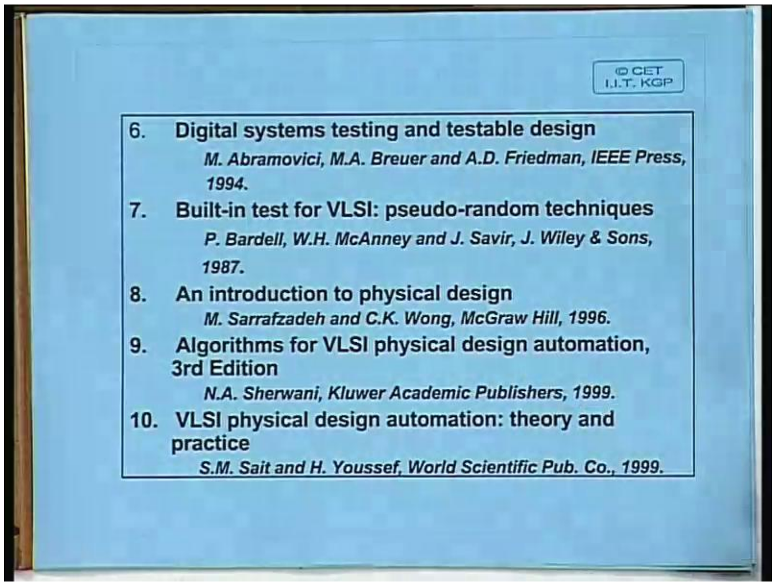
© CET
I.I.T. KGP

References

1. Contemporary logic design
   R.H. Katz, Addison-Wesley Publishing Co., 1993.
2. Application-specific integrated circuits
   M.J.S. Smith, Addison-Wesley Publishing Co., 1997.
3. Modern VLSI design: systems on silicon
   W. Wolf, Pearson Education, 1998.
4. Verilog HDL synthesis: a practical primer
   J. Bhasker, BS Publications, 1998.
5. High-level synthesis: introduction to chip and system design
   D.D. Gajski, N.D. Dutt, A.C. Wu and A.Y. Yin, Kluwer Academic Publishers, 1992.

So I have got a comprehensive list of references. You can have a look at these. So the first 3 references would be concentrating on the design and synthesis aspects. The fourth we would talk about the Verilog description language. The fifth is again synthesis. This concerns high level synthesis. Then we have some more references.

(Refer Slide Time: 04:40)

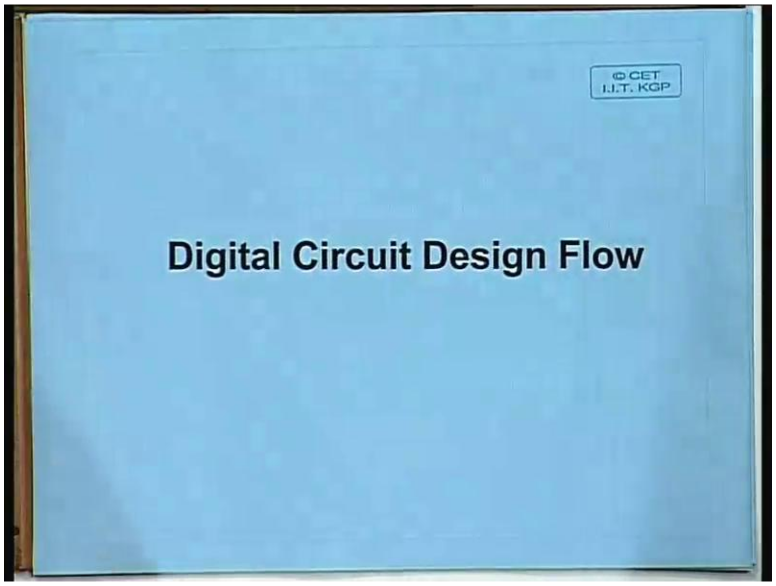


© CET
I.I.T. KGP

6. Digital systems testing and testable design
M. Abramovici, M.A. Breuer and A.D. Friedman, IEEE Press, 1994.
7. Built-in test for VLSI: pseudo-random techniques
P. Bardell, W.H. McAnney and J. Savir, J. Wiley & Sons, 1987.
8. An introduction to physical design
M. Sarrafzadeh and C.K. Wong, McGraw Hill, 1996.
9. Algorithms for VLSI physical design automation, 3rd Edition
N.A. Sherwani, Kluwer Academic Publishers, 1999.
10. VLSI physical design automation: theory and practice
S.M. Sait and H. Youssef, World Scientific Pub. Co., 1999.

There are 2 books which I have referred here for testability analysis testable design and the last 3 books are concerning physical design automation placement layout routing etcetera okay. So during the course of this course we would be we would be referring to these books as and when required okay. So to start with.

(Refer Slide Time: 05:09)

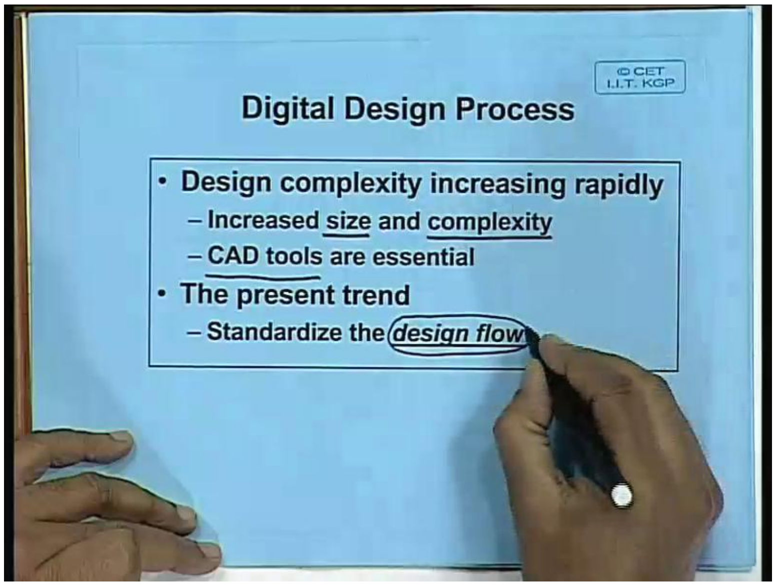


© CET
I.I.T. KGP

Digital Circuit Design Flow

We begin by talking something about digital circuit design flow okay.

(Refer Slide Time: 05:21)



© CET
I.I.T. KGP

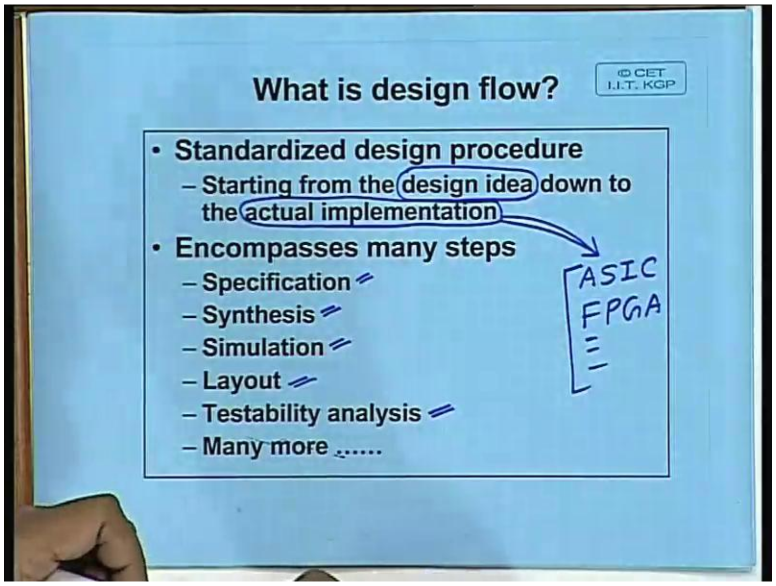
Digital Design Process

- Design complexity increasing rapidly
  - Increased size and complexity
  - CAD tools are essential
- The present trend
  - Standardize the *design flow*

So we look at the digital design process from the perspective of a designer. Now as you know that with the advent of the VLSI technology. Now it is the complexity of design is being increasing very rapidly over the years. Now today it is possible to have chips which can house more than a million trans; multi-million transistor systems are possible on the chip. So design complexity is growing design complexity has 2 you can say 2 dimensions. is of course the increased size and the increasing complexity to handle a design of that size okay. Now in earlier days when the designs were small many of the designs could have been done using some kind of a manual technique or semi-automated technique with frequent manual intervention.

But what has happened today with the increasing size of the designs is that nowadays it is absolutely essential to have computer aided design tools to assist the design process. So the without the computer aided design tools you really cannot handle a design of that complexity. So if you look at the way a designer proceeds with designing a typical chip or a circuit nowadays there is a conscious effort to standardize the design flow. Design flow as I mentioned just previously it is basically the process which you follow starting from the specification down to the implementation. What are the different intermediate steps? Now depending on the kinds of CAD tools you have depending on the kind of chip you are trying to design the design flow can vary quite widely. So it is essential or necessary to standardize a design flow before you are actually trying to design the chips.

(Refer Slide Time: 07:39)



What is design flow?

© CET I.I.T. KGP

- Standardized design procedure
  - Starting from the design idea down to the actual implementation
- Encompasses many steps
  - Specification
  - Synthesis
  - Simulation
  - Layout
  - Testability analysis
  - Many more ......

So this as I had mentioned design flow is essentially the design procedures. Step wise procedure which you follow starting from the design idea or specification whatever is called down to the actual implementation. Now actual implementation when you are talking about well again I have told you. This can be an application specific integrated circuit. It can be a field programmable gate array or it can be any other kinds of programmable logic like PLD’s or any other thing but the idea is this. You have a design idea which you specify in some form and using a sequence of steps using the CAD tools you finally arrive at the actual implementation at the very end. Now out here I show a few of these steps which are furtherer a few more steps also.

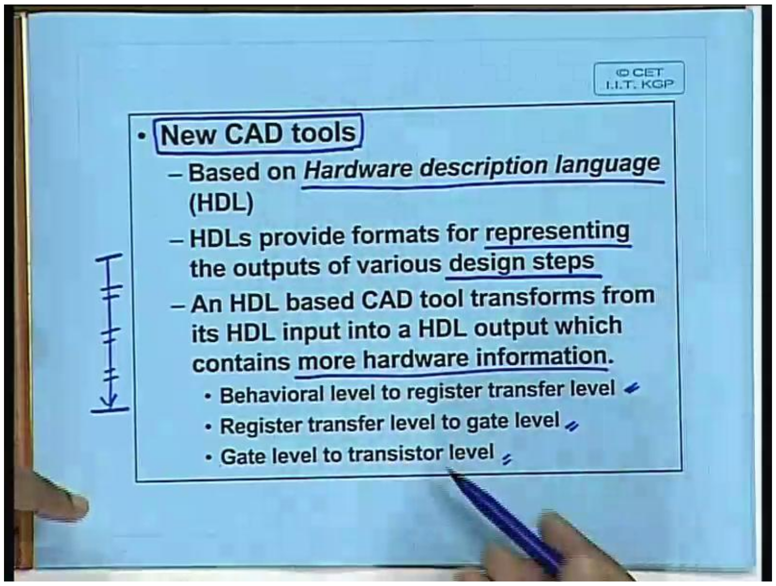
Of course the first step is the design specification. Then you carry out synthesis which is essentially a design translation from the specification to some kind of a netlist. Some kind of a register transfer level or a gate level design. Then you typically carry out simulation to check whether the synthesized design is correct or not. There can be some problem with the CAD tool also. You must be must be hundred percent sure before you actually go for the implementation of the manufactory defects okay. Now the simulation can be done at a number of different levels. This we will be looking at some more detail later because you can also decide to do the simulation after specification itself in order to check whether this specification is correct.

You can do the simulation after synthesis. Well after the synthesis the step after that is typically layout where you are thinking about how to place the blocks you have synthesized on to the floor of the silicon or in case of FPGA. Well how to place or map the sub circuits into the basic cells of the FGPA okay. So these are the layout concerns. Now during synthesis and layout itself you often also consider some issues related to testability. This means when you are designing a circuit well the circuit is functionally working correctly or not that is issue. The design is correct or not that of course is a very big issue but after manufacturing there can be a number of defects. There can be some short circuit in the wire. There can be open circuit.

There can be lot of other problem in the transistors. So after manufacturing you have to test the chips. Now here I have just told you that you could have chips which can house millions of transistors. But externally you can have only a few 100 pins available to you. So just using those few hundred pins how do test those million components inside. There has to be some special design effort that has to go in during the synthesis processes itself so that during testing it becomes easier for the test engineer to test the chip. There are a number of techniques called design for testability built in circuit test. These are the things we will be discussing and there are. Yes. Design processes are such that testability can be made easier.

Okay that exactly what I have said is that designs are big nowadays. So unless the designer puts in some special module some special design you can say design procedures or design rules in the process of synthesis it may become impossible for the test engineer to test the chip. Just if I give you any arbitrary chip and ask you to test it can become a very big problem for you. But if during the design process itself I have incorporated some feature in the chip just using which you

can test it easily. It can become very easy for you for testing okay. Fine. Now as I mentioned that with the increased size of the chips and increasing complexity the CAD tools are essential.
(Refer Slide Time: 12:15)



© CET
I.I.T. KGP

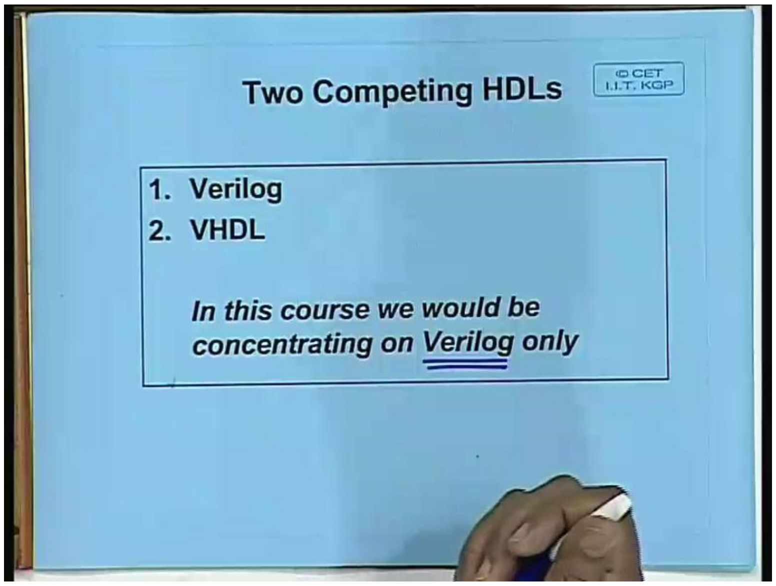
- New CAD tools
  - Based on *Hardware description language* (HDL)
  - HDLs provide formats for representing the outputs of various design steps
  - An HDL based CAD tool transforms from its HDL input into a HDL output which contains more hardware information.
    - Behavioral level to register transfer level
    - Register transfer level to gate level
    - Gate level to transistor level

There is a new generation or a new series of CAD tools which is now available in the market which are used by the designers to go through the digital design flow process. Now all these tools are based on some so called hardware description languages. Well hardware description language means you have a standard way of specifying things. That thing may be your design specification that may be your synthesized net list. That may be the layout. It can be anything but you have some standardized formats in the form of some language. Right. Basically this hardware description language they provide ways of representing the design at several different levels okay.

This we will see later that mean by how. So I have mentioned as we go through the different design steps your design goes on refining from the behavior to register transfer level to gate level to layout. If at each level you can represent the design properly using a description language it becomes a consistent format. Well here advantage you gain additionally is that if you follow some standard your internal representation may be portable across different CAD tools from different vendors. Say there part of a design flow you can do using the CAD tool supplied by vendor A the other part you can do using a CAD tool supplied by vendor B. So if you are following some standard then you have you have complete portability and interoperability.

And typically as I mentioned as you go from top to down from the behavior down to the layout you typically go through a series of transformational steps where each succeeding step will contain more detailed hardware implementation or means hardware information as compared to the previous level from behavior to register transfer level, register transfer level to gate level, gate level to transistor level and so on. So these are some of the typical steps of transformation. Behavior to RTL. RTL to gate level. Gate level to transistor level. Then finally you can have another level. Transistor level to the final layout. Geometries of those layouts the layers okay fine. So let us first talk about the hardware description languages.

(Refer Slide Time: 15:02)


Two Competing HDLs

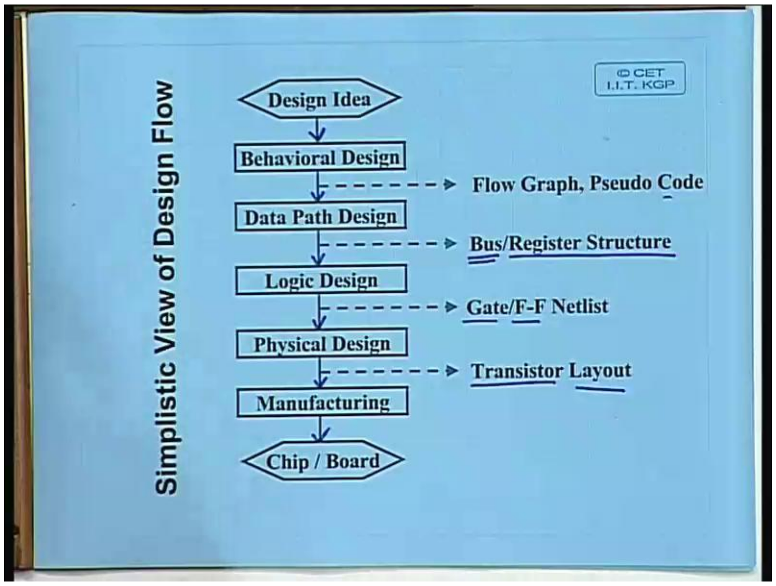
© CET I.I.T. KGP

1. Verilog
2. VHDL

In this course we would be concentrating on Verilog only

Now you may know that today in the market there are 2 competing HDL's which have become very popular is called Verilog, other is VHDL. Now in this course well here we may choose to use any of them. But in this course we would be using the language Verilog for illustrations. Well you can do the same using VHDL also. There is no problem. Verilog and VHDL are 2 different languages with slightly different features. I would be talking about some of the differences that what are the basic differences between these languages. But this I will come little later okay.

(Refer Slide Time: 15:45)

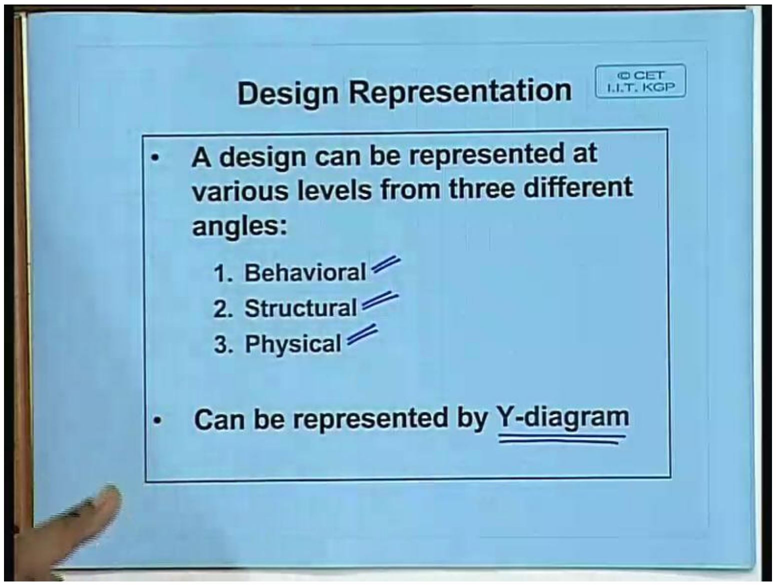


So let me now show diagrammatically a simplistic view of the design floor. Well I am calling, calling it simplistic view because this design flow looks like unidirectional process starting from top to bottom. These are the, this is the direction of the flow. But in a typical design process there can be a few other steps in between and also there can be feedback that in the simplistic flow I have not shown the feedback path. Well here my idea is to emphasize what are the different steps of the process we need to go from a design idea into the final implementation. Okay. So, here as you can see that from the design idea the first thing you try to do is to translate it into a behavioral specification. This behavioral specification is typically in the form of in the form of hardware description language as I told you either in VHDL or Verilog. Then you go through a step of synthesis.

This is step of synthesis where from the behavior you arrive at a so called data path design or a register transfer level design where your basic building blocks are registers arithmetic logic units multiplexers flip flops and so on. So this is a slightly higher level design okay and from the register transfer level design data path design. You typically translate it into logic design where you get a net list of gates and flip flops. Later on from logic design you will have to finally translate it into physical design for you to talk about the transistor their layout etcetera and after you have the final transistor layout you will have to submit the final design specification to the fabrication facility where the chip will get fabricated. This is the process of manufacturing.

So finally after the process of manufacturing you will get either the chip or a collection of chips in the form of a printed circuit board and these are the intermediate forms you can see. After the behavioral design you can get during synthesis something called a flow graph or a pseudo code representing the net list. So after RTL design you get a structure register structure which consists of registers, ALUs, arithmetic units, multiplexers. They may be interconnected by bus buses. Logic design will generate a netlist of gates and flip flops. Physical design, we will talk about transistors and their layouts and so on. So this is the simplicity in view of design floor. (Student Noise Time: 18:51) Data path design is the same as the RTL level design register transfer level design. Yes okay. Now before actually going through these steps let us look at the design process from a slightly different perspective.

(Refer Slide Time: 19:10)



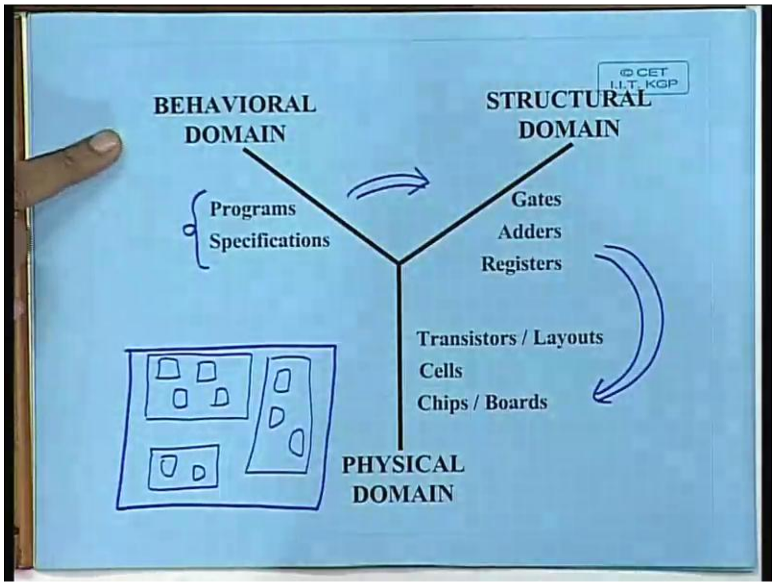
Design Representation

© CET I.I.T. KGP

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

See a design, this design may be in the behavioral level. It can be in the structural level. It can be in the logical level. It can be even in the layout level. But in whatever level you are, now a design can be level can be represented at 3 different levels of abstraction. Well you can view a design from the point of view of its behavior, from the point of view of its structure and from the point of view of its physical implementation. There are 3 different views you have in order to look at a design. Now there is a standard way of looking at these 3 different angles of a particular design. This is represented by something called a Y diagram. I am showing this structure for Y diagram. I will be coming this into detail later when you talk about synthesis. So a Y diagram looks something like this.

(Refer Slide Time: 20:14)



As you can see it looks like a Y with the 3 directions representing behavior structural and the physical domains. Now the idea is that when you are talking about the behavioral domain there we are talking about the behavioral specification of the design. Now here I am not talking about the details later while here we typically specify the behavior in the form of a in the form of description language like Verilog HDL. Well if you if you are talking about logic gate level design there are also you can specify behavior. That can be in the form of a truth table. That can be in the form of a Boolean expression. That does not talk about the actual implementation; just the behavior.

So in the behavioral domain we talk about how this circuit should function. Its input output behavior in some way. Now when we talk about this structural domain this means we have already undergone a step of synthesis. Now we have some kind of a netlist with us. Well if it is a behavioral level then we talk about interconnection of big subsistence big blocks and how they are interconnected. Now if we are at the register transfer level then we talk about ALUs registers adders their interconnection. Now if you are the logic level we talk about gates flip flops and their interconnection and if at the physical level you talk about transistors and their interconnection okay.

Now (Student Noise Time: 22:00). Yes sure. See behavioral level means I am trying to specify the behavior of the design. Now this behavior can be specified in a number of different levels. Well in a high level I can talk about the system. Well I can I can express my behavior in an algorithmic fashion. Something like, you are writing a program in a high level language similar to that. Well the step down I can specify the behavior in the form of say some basic building blocks. Then each of the basic building blocks I am specifying in terms of its behavior like an ALU I am specifying in terms of its behavior.

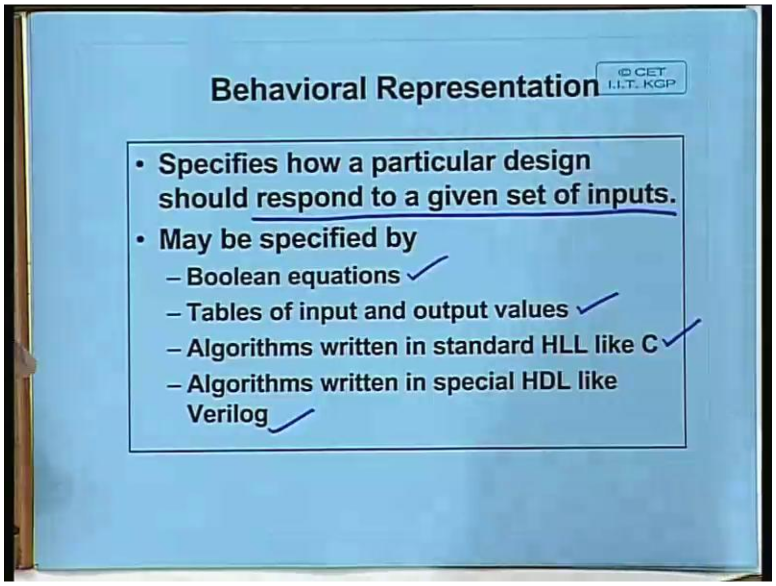
A register I am representing in terms of behavior. I am not going into the detail of it okay and their interconnection. Now at the next lower level I can think of the behavior of a logical level implementation. There I can represent the design in the form of a truth table or in the form of say it is Boolean expression okay. Now this is behavior. Now when you are into the structural domain say at least we have a part of this synthesis which is done. Now we can look at a netlist. Well at the logical level you can see some gates and their interconnections. Well at the RTL level you can see the blocks and their interconnections. Well at the transistor level you can see transistors and their interconnections. That is the structural domain.

Now at the physical domain this is the angle which you look from the point of view of the physical implementation. Like if you consider that this is my silicon floor. This is the floor of the silicon what I want to finally map my design into. So when I am into the high level design step behavior systems of systems there what you can do at the level of the chip you can do some floor planning at sub system I will be placing here, sub system I will be placing here, sub system I will be placing here. But these sub systems are still black boxes to me. Now once we are into level down the specification then each of these sub systems can be broken up into smaller sub systems okay.

Now this process will continue till we go down to the level of the transistors and their layouts. So we will get the final layout at that level. So at the physical domain we are looking at the actual floor of the silicon and how we are placing the blocks which are which are at the level of abstraction we are looking at on to the floor of the silicon. That can be at high level. That can be at a very low level right. So this Y diagram is just a way of looking at it. There is nothing special is there. Okay. So now let us look at some explanation regarding these 3 domains with some examples. Well I have illustrated the differences. Let us now take some examples. Well, let us start with the behavior.

Now as I said that in the behavioral representation we are not specifying the any details of the implementation. We are just specifying the input output behavior of the system. That may be in the form of an algorithm. That may be in the form of a truth table. We found some Boolean expression. So it specifies how to design should respond to a given sets of inputs. If I give some input what is the expected output. That is the behavior right. Now as I mentioned behavior can be specified in a number of different levels by means of Boolean expressions. By means of some kind of truth tables. It can be specified algorithmically in some high level language or in some hardware description language. Well you can specify behavior in a number of different ways. Now let us take some concrete examples of representing this behavior. Well here we take a simple example just to illustrate.

(Refer Slide Time: 26:27)



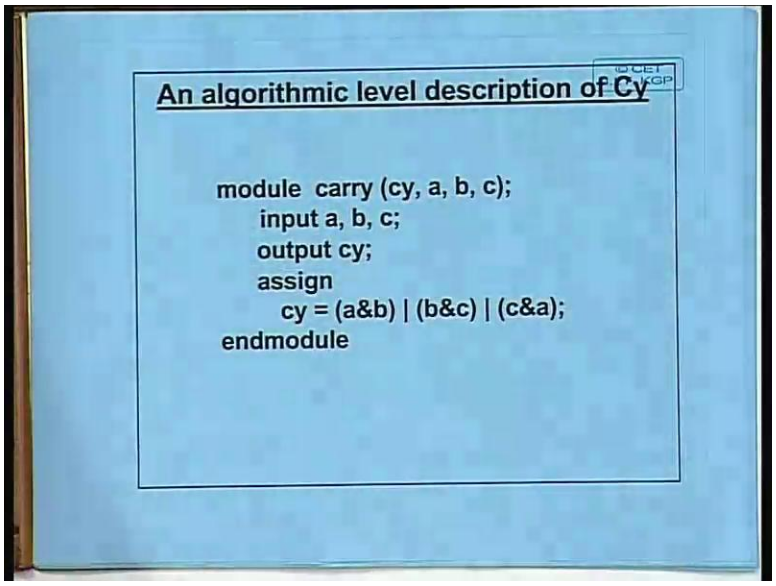
Behavioral Representation

© CET I.I.T. KGP

- 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 ✓

Now here we want to construct an n bit adder by cascading n number of full adders; n-bit adder means full adder. Now as you know that a full adder will be having 3 inputs and 2 outputs. The 3 inputs are the 2 numbers A and B 2 bit C Add and a carry input; and the outputs will be some and a carry out. So as you know that the logic expressions for the sum and carry are these. Right. Well I am illustrating the generation of the carry logic only. You can also do the same for the sum logic. The example that I am giving there I am specifying how I can generate the carry logic for a bit adder. Single bit adder. This can be extended by using a number of such instantiations to create a n bit adder but just by illustrating I am showing you for the time being that how to generate this carry out for a bit adder for a full adder and I am using the language Verilog for illustration. I will be talking about the syntax of Verilog later.

(Refer Slide Time: 27:50)

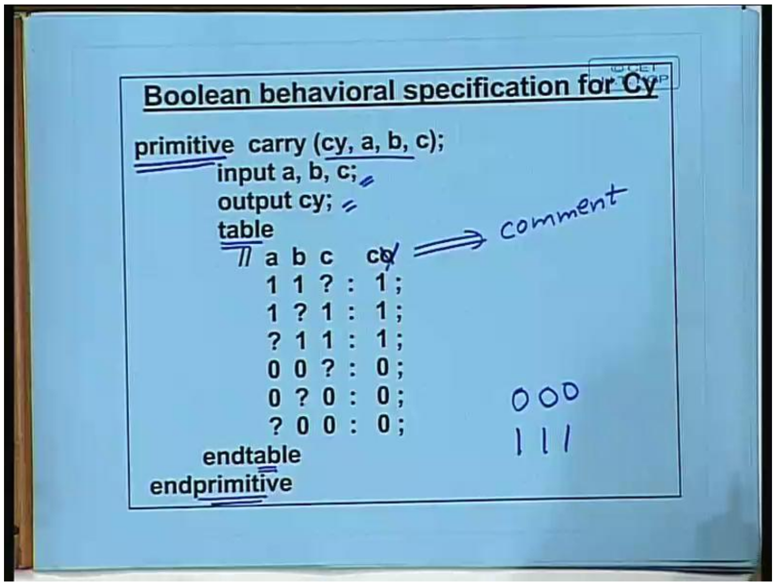


An algorithmic level description of Cy

```
module  carry (cy, a, b, c);
    input a, b, c;
    output cy;
    assign
       cy = (a&b) | (b&c) | (c&a);
endmodule
```

For the time being I am showing you how to specify. This is an algorithmic level description of carry where as you can see here I have specified the Boolean expression. This ampersand means and this bar means or so ab or bc or ca. This is how we specify this in Verilog. This is a module. In Verilog everything is in terms of a module. This is a name of the module. These are the parameters just like you specify a function. These are the input parameters. These are the inputs to the block. This is the output, this is a keyword assign. This is assigned to cy okay. So this is this is way of specifying the carry in terms of its behavior okay. There is an alternative way of specifying it also in a in the form of a behavior namely in the form of a truth table that you can do like this.

(Refer Slide Time: 28:54)



Boolean behavioral specification for Cy

```
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
```

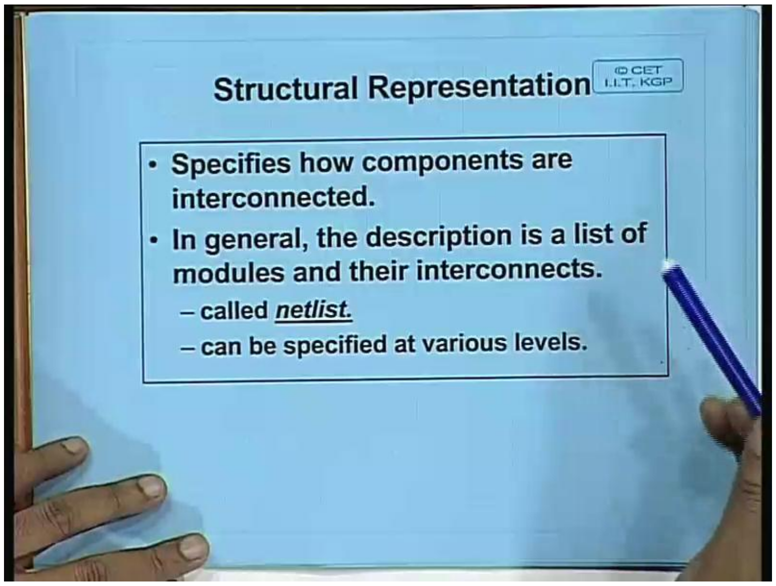
comment
000
111

So here well when we specify it in in the form of a truth table instead of module we declare it as a primitive right. Begin primitive end primitive some kind, Well, the rest are similar here. These are the parameters. These are the inputs. These are the outputs and the input output behavior we specify it by using a keyword table and end table. Now this table we will remember that how many inputs I had and how many outputs I had. There were 4 double slash means comment. This is a comment line. This line is nothing but a comment line. Just look at the other lines. First you specify the inputs, then a colon, then the output followed by a semicolon.

So this can be an incompletely specified truth table with you can see are these are question marks which means do not care. This says if a and b are c is a do not care. Then the carry out will be sorry this will cy. Carry out will be one. Similarly if a and c are b do not care then it will be one. If 2 of them are zero then the output is zero. So you can see we have specified all possibilities but we did not specify all eight column all eight rows of the table okay. Because the all zero case will get covered. By these, the all case will get covered by the first 3 right. Fine.

Yes. (Student Noise Time: 30:28) Primitive is used for something which is like a constant for you to specify exactly the input output behavior in terms of the numbers or the values. Now in the early example you have specified in the form of Boolean expression where you need to evaluate the expression in order to find out the value. But in the case of primitive it is something like constant values you specify, that if this is input this is the output okay. Fine. So these are some examples of representing a design by its behavior. Now let us talk about structure.

(Refer Slide Time: 31:10)

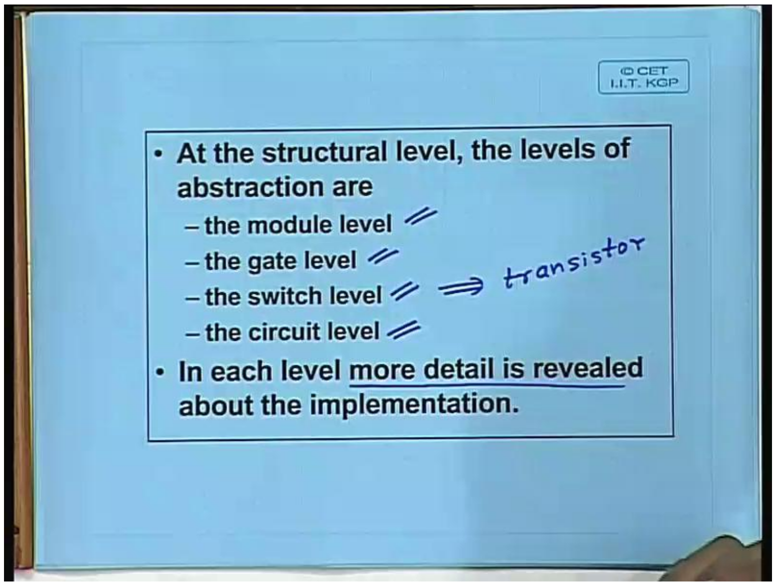


Structural Representation

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

Now in the structural representation here we have some components. Now the kind of the components we have that depends at which level of design we are in. Now if you are at the logic level or components or gates or flip flops. If you are at the register transfer level or components or ALUs adders multiplexers etcetera. So here you specify components and how they are interconnected. This interconnection is sometimes called a netlist. Netlist is nothing but a graph. You specify some nodes and their interconnection. Now these nodes will be the components. This will be gates. Okay. So now as I mentioned the netlist can be specified at various levels depending on at which level you are considering the components. Components can be as low as the transistors or the gates or it can be as high as say an ALU or even a subsystem. Okay.

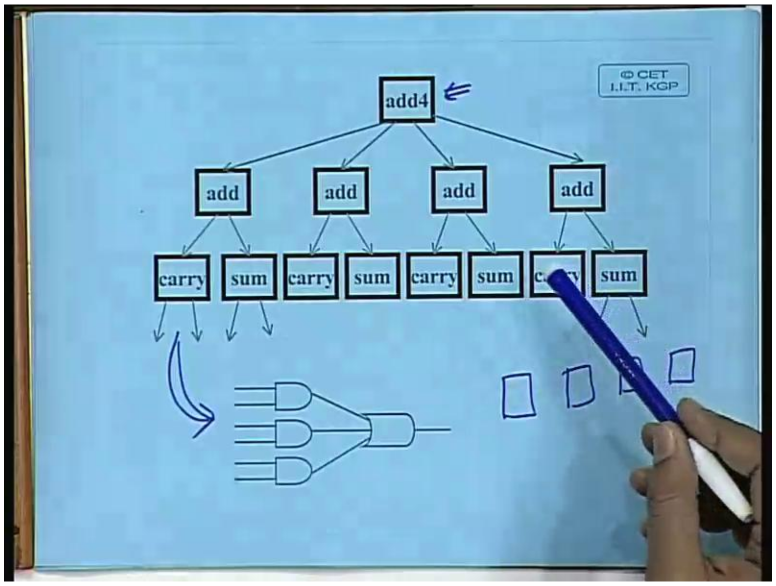
(Refer Slide Time: 32:30)


© CET
I.I.T. KGP

- At the structural level, the levels of abstraction are
  - the module level $/\!\!/$
  - the gate level $/\!\!/$
  - the switch level $/\!\!/ \Rightarrow$ transistor
  - the circuit level $/\!\!/$
- In each level more detail is revealed about the implementation.

So, just as I mentioned this structural representation you can you can represent it at the level of modules at a very high level. Modules can be very high level. It can be RTL level modules also. You can represent it at the gate level. Switch level means a transfer level. This switch and transistors are used synonymously. So when you are talking about switch level design you are actually talking about a netlist of transistors. But when you have translated this transistor level design into actual physical layout then you are at the circuit level. It is very clear as you go from the module level down to the circuit level more and more design details are getting revealed. Okay. Okay. Now we will illustrate a very simple example.

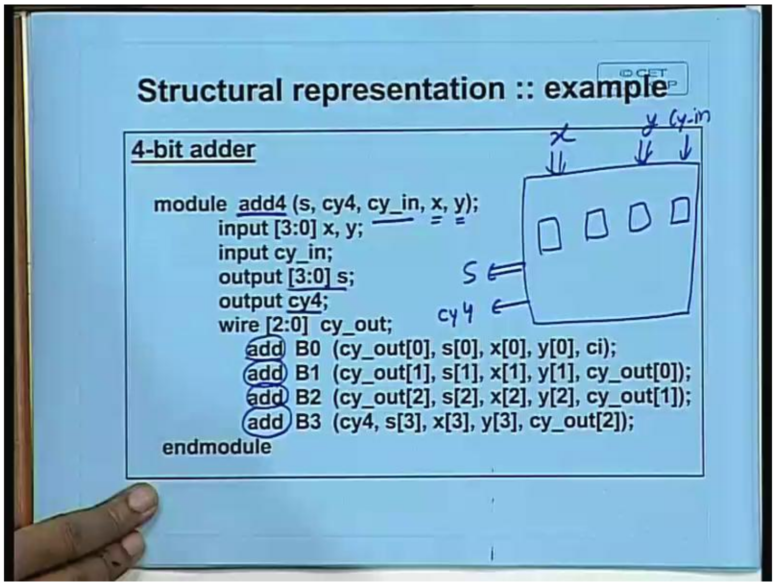
(Refer Slide Time: 33:36)



Say we want to design we want to design a 4 bit adder. The root of this tree represents that specifies at 4 represents a 4 bit adder. Now to keep things simple we are designing it as a ripple carry using 4 such we will be using 4 full adders and interconnect them. Okay. By rippling the carry, so the 4 bit adder can be broken down into 4 full adders and their interconnection. So here we are talking about structure level. So at the first level of abstraction you can break it down into 4 full adders and their interconnection. Well if we if we want to go down further then each of these full adders can be broken down further into a carry sub circuit and a sum sub circuit.

Now each of these carry and sum sub circuit can be further broken down into an equivalent logic level. Just as for example the carry sub circuit looks like this ab or bc or ca right. Similarly you can design a sub circuit for some. So this is your structural level design. So as you go down the size of your the building blocks at this level it were full adders at this level it were the carry and sum blocks. At lower level it was the gates. So you have smaller and smaller building blocks and the netlist is becoming bigger and bigger in terms of the number of components okay. So this kind of a design specification you can represent very easily in a hardware description language. I am giving an example with Verilog. Okay. This is a design description using Verilog.

(Refer Slide Time: 35:32)



Structural representation :: example

4-bit adder

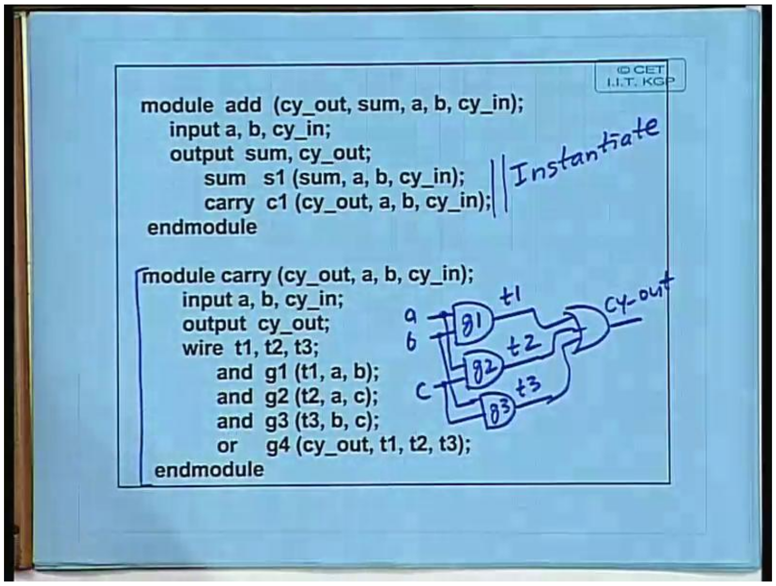
```
module add4 (s, cy4, cy_in, x, y);
        input [3:0] x, y;
        input cy_in;
        output [3:0] s;
        output cy4;
        wire [2:0] cy_out;
           add B0 (cy_out[0], s[0], x[0], y[0], ci);
           add B1 (cy_out[1], s[1], x[1], y[1], cy_out[0]);
           add B2 (cy_out[2], s[2],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

x
y
cy-in
s
cy4

If you look at it you will understand what you mean? This is a module which is a 4 bit adder. We are giving it a name add4. These are the parameters. This is the carry in. These are the 2 inputs we want to add. Now since this is a 4 bit adder the inputs will have to be 4 bit getters. So as you can see this x and y are defined as input within bracket 3 colon zero. It means it is a 4 bit vector with the indices starting from zero going up to three x 0, x, x 2, x 3 and y 0, y, y 2, y 3. Similarly carry in is a single bit. The sum will again be a 4 bit quantity and cy 4 is the carry out. Okay. These wires, these are some intend say. This is a block you are considering.

You have a vector y. You have x and a vector y. You have carry in and at the output we have a vector s and we have a carry out cy 4. Now inside well see just look at these 4 lines. Here what we are doing? We are we are creating 4 instantiations of this low level add module. Add is a full adder. So what we are saying that in this block we are instantiating this add 4 times right. These are the names we are giving b 0, b, b 2, b 3 and just before coming back to this let me show you another slide first. This adder this adder must be predefined as a module.

(Refer Slide Time: 37:33)



```
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 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
```

© CET
I.I.T. KGP
Instantiate
a
b
c
g1
g2
g3
t1
t2
t3
cy-out

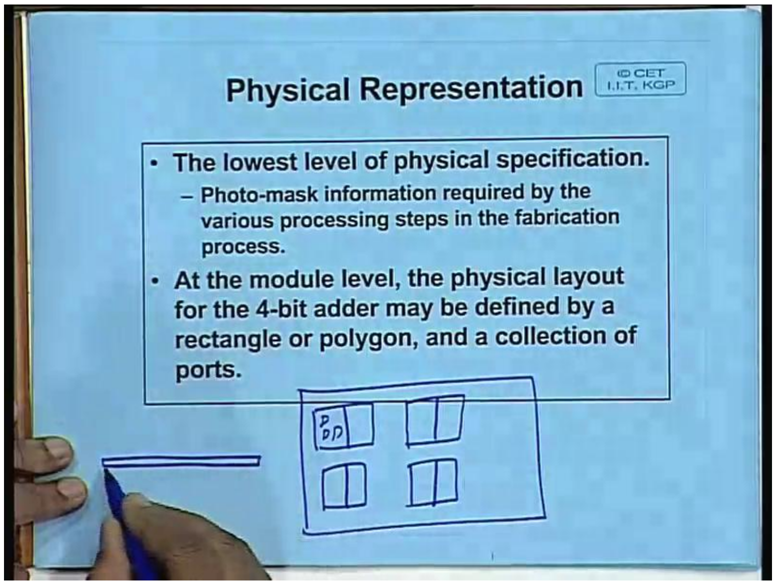
As another module, this is a full adder module which I have already defined where the first parameter represent the carry out then sum then the 2 inputs and the carry in. Right. And similarly the carry see here this adder if we just look into the detail here I have not specified the Boolean expression. I could have but I have decided to use a separate module for carry and a separate module for sum and use it here. These are examples of instantiations. Instantiate just like we are calling a function from another function we are just including a module inside this module. So we are including the sum. We are including the carry and we are using the parameters in a proper way so that these parameters we can get proper correspondence.

Well here I am showing the quote for carry. For sum I am not showing, sum will be very similar. Carry here at the lowest level you have a choice of specifying the circuit in terms of gates or you have a choice of specifying in terms of behavior in terms of Boolean expression. Well here this and or are primitive gates which are already available in the library. These we need not explain any further. These are already there. So if you look at this and t a b. This means g is an AND gate. It inputs a and b and output is t. Similarly g 2 is another and gate. Inputs are a, and c, output is t 2, g 3 is another, b and c are the inputs, output is t 3 and g 4 is an OR gate. This is a 3 input OR gate. It takes t, t 2, t 3 as the inputs and at the output it generates carry out.

Now as you can see this t, t 2, t 3 were not any input or output parameter. They are intermediate lines. These are defined as wires right so wires are essentially the signal lines which are used for interconnecting module with another internally. So this is how you can define a carry a full adder. And using this and their interconnections you can at a higher level you can define 4 full adder modules and then interconnections. See interconnections are defined in the way you specify these parameters. Just you see b 0 has this cy out 0 as the carry out, b has this cy out 0 as its carry in. So the carry is getting changed. Similarly this cy out is getting as the carry in here. The carry out of this is getting as carry in.

So this 4 full adder modules we are doing a repeat change by specifying parameters in a proper right. (Student Noise Time: 40:52). This is at the RTL level. Yes. So may be at the top level of the design you do this. But as you go down the design you will breaking up this adds into this. At the next level you will be splitting this sum and carry into this. So this step goes on but the beauty of this language verilog is that you can capture information in the same language. So the same language you can specify things at a number of different levels. There is a version of verilog using which you can also specify the physical layer. I will just show an example. Fine.

(Refer Slide Time: 41:38)



Physical Representation

© CET I.I.T. KGP

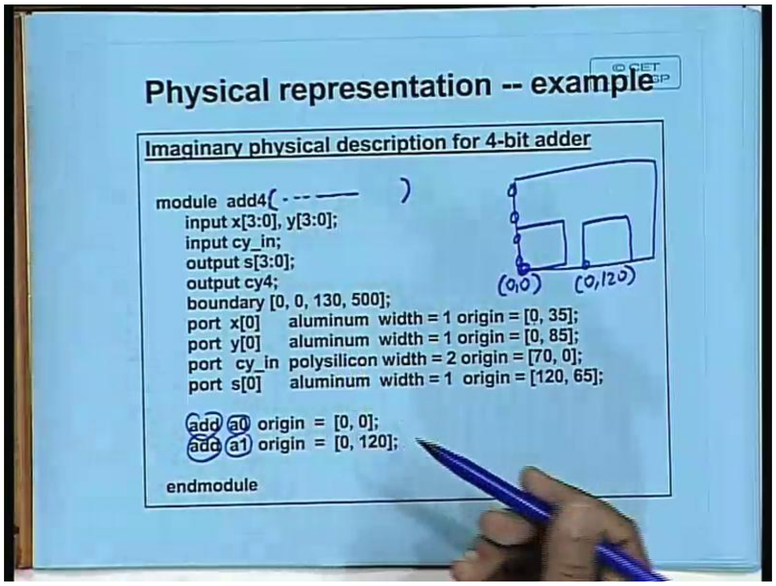
- The lowest level of physical specification.
  - Photo-mask information required by the various processing steps in the fabrication process.
- At the module level, the physical layout for the 4-bit adder may be defined by a rectangle or polygon, and a collection of ports.

So the third step that I was talking about was the physical representation in the Y diagram. So we talked about behavior. We talked about structure. We talked about physical. Well here just before I proceed further, let me tell you thing that if your objective is to design a chip. Then it is always instructive and it is always better to design or to specify in the structure level as far as possible because you may be tempted to specify in the form of the behavior because all of you are so very familiar with writing a program in a high level language. So if I give you a complex design to implement then you may be tempted to write it like you write a program in c.

But the problem is that the synthesizer while synthesizing can face a lot of problems. But if you have some kind of a block diagram hardware block diagram in the back of your mind you can specify it at the structural level to the extent possible. Then you can get an implementation which will be efficient. Well if you specify the behavioral level most likely you will be getting a very inefficient implementation right. Fine. So in the physical domain we talk about physical representation and there I had mentioned with the help of an example earlier that when you are at the module level when you are at the RTL level then at the level of the silicon floor we can represent each of the 4 full adders as rectangular regions or polygons. These are the 4 full adders.

As you go down, as you go to the carry and sum then each full adder will be divided up into a carry and sum, carry and sum, carry and sum, carry and sum. Then at the logical level each of this will be divided into gates okay. So these steps are going on and in the physical representation at all levels you are thinking that exactly where on the floor of the silicon I am going to map that. And finally when you are at the layout level at the lowest level then you have the actual layers metal poly silicon diffusion etcetera. Now the ways in which the lowest level layout is represented is also not very difficult. They are represented as some geometrical shapes. They are all rectangles or a collection of rectangles. A line can be defined as a rectangle whose width is less but length is long. Who specify the exactly code names just like you specify a drawing. So I am giving you an example.

(Refer Slide Time: 44:33)



Physical representation -- example

Imaginary physical description for 4-bit adder

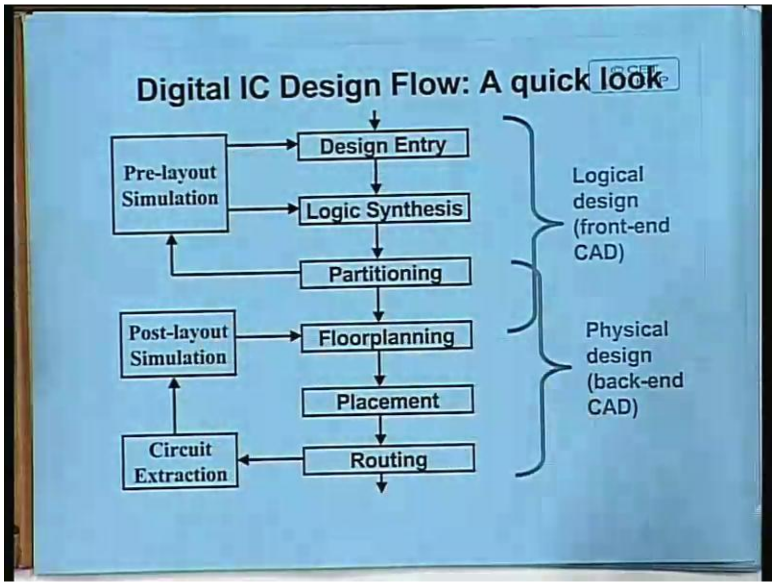
```
module  add4( - -- ——    )
    input x[3:0], y[3:0];
    input cy_in;
    output s[3:0];
    output cy4;
    boundary [0, 0, 130, 500];
    port  x[0]     aluminum  width = 1 origin = [0, 35];
    port  y[0]     aluminum  width = 1 origin = [0, 85];
    port  cy_in  polysilicon width = 2 origin =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

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

endmodule
```

(0,0)
(0,120)
© CET IIT KGP

This is of course a hypothetical example. This is not the exactly I am only showing you a small part of it that how a physical representation can be moduled in Verilog. That means here also you have the module. Well just forgot to add the input and output parameter. They will be as usual okay. The input and output. Boundary specifies the coordinates of the bounding box inside which you want to place the layout 0 0 to 30. These are the xy coordinates and these are the ports through which exactly where on this bounding box are the input output points located. These are called the input and output ports. So you specify the exact coordinate of this and you also specify on which layer the ports are available. For example this is available on metal aluminum width of. This is available on poly silicon width of 2 and there can be some other sub modules you may be adding.

A zero, for a zero can be anything. Sorry sorry not a zero add. This adds are full adders. You can add full adder at origin 0 0. So you add here. If this represents 0 0, then you can add full adder here and another full adder you can add at a coordinate 0 20. It can be somewhere here. This can be 0 20. So at this level you specify the exact coordinates and you specify at exactly at which coordinate you have the blocks. Now these blocks can be high blocks. It can be in the form of rectangles of aluminum poly silicon diffusion also. Okay. So here we will not be going into the detail at this level okay. Just for your information I have mentioned this.

(Refer Slide Time: 46:37)



You should summarize digital IC design flow. These were the steps some of the steps I have mentioned. This is a slightly refined diagram. Just I will finish this lecture with this slide. See to begin with you do design entry where you specify your design in terms of its behavior. Then you go through a process of synthesis. This is sometimes called logic synthesis because the RTL level synthesis and logic synthesis are they are typically combined together. So there what you get as the net output is a netlist of gates. Okay. So after you get it you typically do some kind of a circuit partitioning because your overall netlist may be too big.

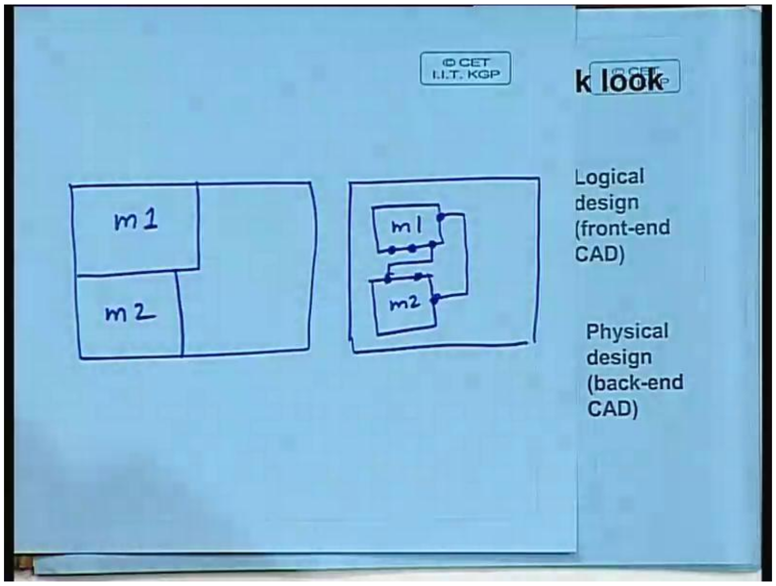
Well out here you do some kind of a graph partitioning to identify the tightly coupled modules and the number of interconnection between 2 modules or less so that it will be easier for you to place these modules on the floor of the chip. Typically either after partitioning or before partitioning, you do something called pre layout simulation to verify whether your behavior or the process of synthesis was correct or is there anything wrong. Now you will see that if you specify a design in verilog there are some features which are perfectly okay when you specify it for during synthesis it will be better it cannot be synthesized. These are the things we would be discussing later. So you carry out a pre layout simulation and if you get any errors or inconsistency.

You can modify either your original design specification or you can make some change in logic. Logic synthesis also in order to rectify the defect. After you are done with this you are now going to the actual layout implementation on the floor of the silicon. There you actually go through processes like floor planning placement and routing. So at the end of which you get the actual layout in terms of the rectangles that I have just mentioned, silicon poly silicon diffusion metal etcetera. But as you know the process of fabrication is expensive. Well if you give a chip for fabrication it will cost you. It may cost you several lakhs of rupees. So before going there you must be sure that your layout is also correct.

So normally you go through a process called circuit extraction. Circuit extraction means from the layout from those geometries you would try to re extract the transistor level circuit. Like for example if a polysilicon and a diffusion rectangle intersect it means there is a transistor at the junction. That way it is very easy to extract the transistor level layout from there and means you can also estimate a very accurate figures of the interconnect resistances and capacitances because you know the exact geometries. Now say there for a poly silicon line you know the resistance for unit length. You know that exactly for how long the line is running. You can estimate the total resistance.

So after circuit extraction you can do a post layout simulation but the problem here is that here you have a circuit which consist of transistors resistances and capacitances. So here you will have to do something called a circuit simulation using software like spice or something. This is time consuming. This can only handle small circuits. So you will have to use this very judiciously. Yes. (Student Noise Time: 50:39) Floor planning means you floor planning is a tentative plan that exactly where you want to place the blocks. Placement means as you go through the step of synthesis. Say what exactly mean to say something like this.

(Refer Slide Time: 51:00)


© CET I.I.T. KGP
k look
m1
m2
m1
m2
Logical design (front-end CAD)
Physical design (back-end CAD)

Suppose this was the floor of the chip. So at the level of floor planning you have defined that this is a place where you will be placing a module m. This is a place where you will be placing module m 2. This is just a plan, just like a building plan you have done but as you go through the process of synthesis further you will be going down to gates to transistors to layout. There you will be getting the exact shape of the module. So the after the placement is over possibly your modules will look like this. This will be now your m. This will be your m 2 and these will be the interconnecting ports. So you will be now interconnecting them like this.

So floor planning is a tentative relative ordering or positioning. And placement, well you can do placement once you know the exact geometries and the exact points which need to be interconnected. Okay. (Student Noise Time: 52:22) Sir routing will be after placement or routing will be done after placement. (Student Noise Time: 50:30) Routing of course will give you all the details of which layers are connected. Yes. But typically when you are connecting modules at a higher layer you just try to keep most of the interconnections at the metal layers. But for some connections are shorter distances you may also use other layers. (Student Noise Time: 52:59) How can we do post layout simulation? Post layout simulation. See you may not be doing it frequently.

But it is a very good practice because I am giving a very good example. You have done a design. You do a simulation at the high level without going through post layout and you may see that your circuit is working at gigahertz. You are very happy but you but you get your chip fabricated it comes back you see that your circuit is working at 15 megahertz only. That is the fact of life. You have ignored capacitances and inductances the noise cross talk everything you ignored but once you do a post layout simulation. You can find out the problem areas of the circuit where the delays are occurring. You can possibly redesign that part so that bottle necks are removed is placement only we are deciding which interconnection we do. (Student Noise Time: 53:49)Placement we are deciding the geometries of the blocks. The interconnections are done in the routing phase.