

Design Verification and Test of Digital VLSI designs
Prof. Dr. Santosh Biswas
Prof. Dr. Jatindra Kumar Deka
Indian Institution of Technology, Guwahati

Module - 1
Introduction
Lecture - 2
High Level Design Representation

A welcome to the second lecture on module 1 which is on High Level Design Representation. So, in the first lecture, that is in the last first lecture of this video course that is in the last lecture. What we have seen? We have seen that actually in this course what will be studying, we will be studying about how we can develop or algorithms which are required to develop some automatic tools, that is CAD tools which will help us in design of digital VLSI circuits.

So, I mean if you already we have told you in very brief that I mean if we talk about digital VLSI industry, so I mean there are basically two paradigms; one is the design paradigm and another one is the cad tool paradigm. So, what do you mean by design paradigm? So, that in this case we take some specification then we design a circuit, finally we fabricate it, test it and sell it to the market. So, depending on user specification like it may be a processor or it may be a embedded processor for mobile it may be a video processing circuit, it may be a image processing circuit, it may be something for bio medical image processing.

So, depending on different requirements of some customer or some specifications of or some devices or some of the applications, we can develop hardware for that. Actually this is similar to software development, but we already have seen that hardware are more if you represent some algorithms in hardware, they are more fast, they are more comfortable because size is less, their power is less and there are so many advantages if you convert a hardware I mean software algorithm to hardware representation. So, that is why we slowly moving towards hardware representation for all most all the algorithms which we could improve I mean which we were I mean traditionally we have a software version.

So, we are moving towards hardware version. So, that all things are faster they are so smaller in size and their low power and so forth. So, I mean that is of the design

paradigm. So, given a specification how can we design a good hardware for that? So, there is another paradigm which actually the cad tools. Now as also we discussed that there are designs are becoming so and so complex these days.

That as a human being you cannot, you can just write the specification then you can may be write a very log code that is you can represent your specification in some language, but after that you have to go for the high level design, then you have to go for your gate level design, then you have to go for backend design and then you have to generate the test patterns, then you have to go for fabrication and so forth. So, all these steps are actually so difficult or so complex in nature.

In case of digital design mainly so it is impossible for a human being or even for a group of people to do that manually. So, for that what do you do? You require cad tools. There are actually complicated tools are there. So, which will take your specification and then to convert it into high level design and again it will convert it into gate level design and it will converting into layout, then it will generate test patterns, so all the phases have fully automatic these day or can say almost automatic these days or very little manually intervention.

And then actually what happens? So that you get totally a manual I mean totally an automated flow and finally, you get your chip. So, now the second paradigm that is the cad tool paradigm is that the designs are becoming more and more complex. So, the complexities are most of the algorithms as we slowly see in our course are all NP-config and NP-hard problems.

That is there is no well known polynomial time solution for that. Say, for example, if there is a if we want to do the gate representation for a circuit so there can be say the input problem is of order n so number of elements or inputs may be of order n so what are the different types of circuits possible to represent these circuit in the forms of gates will be actually in a exponential order. So, you have to select the best one. So, while we slowly when we go down in the line of course, we will see that most of the problems if the input space is of order n , the output or the number of output of the complexity involved to compute the solution to those problems will be exponential. So they are actually non polynomial time, non deterministic polynomial time.

So, let us not go into the details of the theoretical computer science here. So, the idea is that there is no well known polynomial time solution for most of the algorithms for most of the problems in digital design cad flow. That is for example, if you are like if you have given a specification I have to verify whether the same say given your specification now you have designed a high level circuit for that, high level representation for that. Now you have to verify what you saw that by formally, by verification methods that this output that you have generated is a high level design is similar that it satisfies your input specification, because you are going from one form of representation to the other.

Similarly, from the high level design you can have a gate level design. So, from high level design to gate level design you can go for transformations. So, these all what we have discussed in the last lecture this stuff. So, it says that when we go from one transformation to the other transformation they should be equivalence. So, equivalence should be preserved that is they should satisfy the initial specification. Then you should do that formally. So, if you verify that formally or if you verify by applying all possible input patterns so the problem order is exponential. That is if n is the number of inputs the vectors to be tried or number of times the algorithm will run or number of steps or iterations in the algorithm to verify them etcetera will be in order of 2 to the power n or even higher.

So, all the problems are highly complex. So, you cannot have polynomial time solution for that. So, we have to find out good heuristics or good simple algorithms which will give you an approximate or a near or a solution which is very near to the optimal solution, but may not be exactly, but it will be around 99.9 percent accurate and 99.9 percent near optimal solution using some heuristics method in which whose I mean complexity is much, much less than the exponential complexity. So, all these things will see when you will go through the course.

So, the basic idea was trying to say that the cad tools the second paradigm of digital design that we have to design good algorithms which will actually help you in developing an automated which will take your specification and automatically gives you a circuit. For all the intermediate problems which we have discussed in the VLSI flow high level synthesis, gate level synthesis, then following that layout verification test pattern generation, all these problems are NP-computer exponential ordered problems. So, you have to develop good algorithms which will actually solve the problem and will

not take a very huge complexity. That is it will be much less complexity exponential order for most of the cases or there will be having very less complexity compared to exponential order.

But at the same time, you cannot expect a hundred percent optimal solution as in the case if you are running an exponential algorithm, but your solution even if your low end algorithms or low complex algorithm for the same problem will give you a very near accurate solution or the complexity is lower, but the accuracy or the near optimise result of the solutions will be very near to the most optimal results or the optimum results which you will or the best result I should say.

The best results or the optimum results we could get by the older algorithm which is actually the exponential order algorithm, say the result or the optimal result say some, x y z kind of thing. So, your solution that you will get by the low end algorithms low complex algorithms should be very near say that is a 0.99 times the real values of x y z something like that. So, that is why now through this lectures or lecture series I mean in the design part, what we will study how you can develop, your first show for most of the problems we show that they are NP-config problems. So, they are very hard problems to solve because there is exponential complexity. Then will shows that how can you develop some heuristic algorithm. That is the complexity will be lower that is you can get a near optimal solution for them. So, that will be the basic key idea for the lecture series in the design module of the course. So, slowly we will go for step by step.

So, first level what we have seen is the high level synthesis that is we have a given a specification .Then you have to convert it into a architectural block level design. So, that is actually called the high level synthesis. So, that is the second level second lecture is on high level design representation. So, our first step will be to how first we will show that the problem is the very complex problem that is given a specification how can you develop a high level design or architectural design for that and then we will show how can we develop tools which will you can automatically get you the high level design or the architectural design for that and yet the solution will be very near the optimal solution. But the complexity of the algorithm will be much, much lower. So, that will be our basic notion in this.

(Refer Slide Time: 08:06)

Introduction

- Almost all steps of VLSI design are automated.
- Any automated procedure requires that input data being provided is in some predefined format. Also, the models used to represent the inputs and transformations (changes of the input) should be efficient for execution of the procedure.
 - For example, in case of HLS the input specifications are generally in some Hardware Definition Language (HDLs) like Verilog, VHDL, System C etc.
- The HDL specifications are represented using several modeling paradigms like Control and Data Flow Diagram (CDFG), DeJong's hybrid flow graph, SSIM flow graph, Finite state machine with data etc., which are suitable for scheduling, allocation and binding procedures.
- Sometimes timing constraints (on execution of steps) are also given in the specifications, which are modeled by the above paradigms, however, with timing parameters included e.g., CDFG with timing, DF with timing and CF with timing.

NPTEL

So, let us see for the first step that is the high level synthesis, so as we have already discussed almost all the steps of VLSI designs are automated. So, again now in the next series say that whenever we have an automated algorithm, so you have the algorithm will take some input and will generate some output is almost true for every algorithm. So, even for the searching algorithm or shorting algorithm, there is an input and you to generate an output.

So, for all the algorithm input, inputs are predefined. Like say for example, if you want to go for a search solution, search of integers. Then what is the input? Input patterns are you have to give some n or n or k or whatever number of elements, it will be some integers and then you have to give a key like say you have numbers like 1, 10, 15, 25 and 30 and the key is a 29. And they have to find out whether the key is in the list. That is actually called the input format of the algorithm.

Similarly, for any automated procedure requires that input data is given in some predefined format. So, in case of circuits so like in case of shorting problem is very simple problem or searching problem is a very simple problem. So, you have given an array of integers and a key. But here you has actually the specifications are very difficult or it will be very complex in nature like, the specification can be a addition of two numbers like you say calculated, addition, subtraction, multiplication, division or it can be as complex as a processor.

So, processor will have keyboard interface, output interface, screen interface, it will have image processing, it will have instructions for everything all of you have gone through your undergraduate course on computer architecture. So, you can know how complex this specification of a processor. So, complexity of specification can be as simple as that of a calculator or it can be as complex as that of a processor. So, I mean in case of very simple algorithm like searching and sorting, input specifications are very simple. So we do not say a very formal notion for that. But in case of circuits, as the input specification format or input specification is very, very complex, so we define some formal models by which we can represent our specifications.

And while so for example, and also the input specification that is a use some language or use some models to represent your specification and this should be also, efficient for I mean implementation of the procedures. So, what does it mean? It means that first of all you should be having a unique or I mean a standard kind of representation for the specification or some models for the specification and now your algorithm or the cad tools will run, which will take the inputs and generate an output like it will take an input specification and generate the high level architectural block diagram. Right, that is what the idea.

Now whatever model you use to represent or whatever language you use to represent that should be a very we need to say it should be quite amiable or it should be quite easy or it should not be very complex for the procedure or for the algorithm which will convert the specifications to architecture design to use them. Or in the other words, the models should be very much useable or should be low complex and useable, but the algorithm which will convert this specification to architecture design.

So, if our input specifications or output specifications which will be done by the algorithm so the models are the input specification input representation will be such so that, they can be used very easily or in a very low complex manner by what do you call your algorithms. So, that is why they are saying that the input should be efficient for the execution of the procedure.

For example, in case of high level synthesis, what is your input specification? Input specification we represent in some hardware definition languages which are called HDLs hardware definition languages and some examples are Verilog, VHDL, system C

etcetera. So, in this course we will be looking into Verilog. So, it is very, very similar to C or with some small differences I mean like which represents the hardware. So, we will see that when you will go through the course. There are many languages like Verilog, VLDL, system C, system Verilog and all. So, there are so many languages are there. So, that we use them to represent your input specification like for example, you want to say that I want to implement a hardware, which will actually say, will add some n numbers or short something and all. So, writing in this language specification is always full of ambiguity.

So, if you write it in standard language like Verilog or VLDL, which are called the hardware definition languages. Then they are standard, there is no chances of any ambiguity, and it is very standard representation and very easily executable by the all the procedures which will be used for converting specification to high level designs. And also, from the somewhat do you call those hardware I mean the Verilog and VLDL are nothing but some language. So, the language must be represented by some formal models like for example, we know that so if you want to represent some travelling sales person problem or you want to represent or say some problem like n cities travelling sales person problem from one node to another and so forth.

So, always map them to a graph like say say most of the problems can be mapped to a graph like, this is the one good example like, travelling sales person problem. We map them to a graph and if there is a path where the cities are represented by the nodes and where there is a path from one city to another city and two nodes by an edge, then the distance between the two parts or two cities can be represented by the weights on the edges. So, that is what? That is nothing but we are representing a problem in terms of formal model or in terms of a graph model.

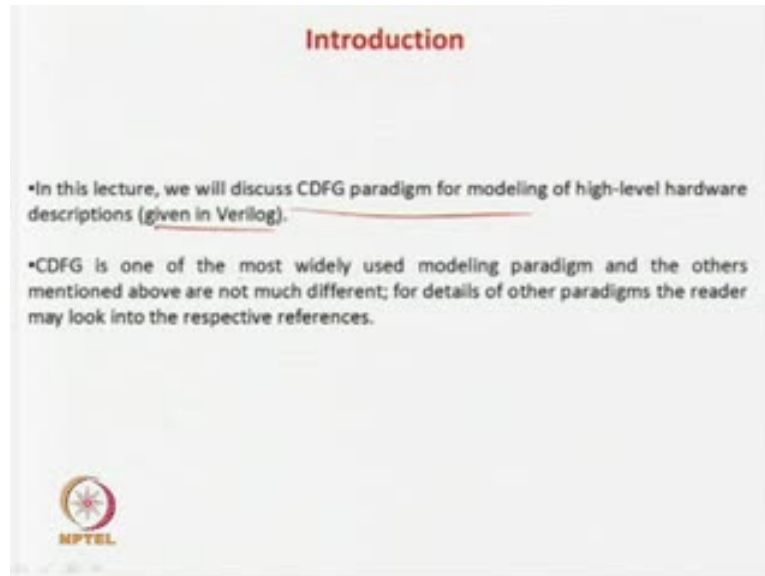
So, this is a well known thing in the theoretical computer science and data structure algorithms. Mainly most of the problems we try to represent them using some data structures like graphs, trees like a binary search trees so some sequence of numbers we represent them in the form of what you call as tree. Then some nodes are there then some left child is there, right child is there. If the root has some value if the left child is having node is having less value less value than the root then we put it to the left side and if greater the value then we put in to the right side. So, all this things we have already learned in the undergraduate course.

So, what I am saying is that here problem is here representing the graphs and trees and so forth. So, there are some formal models which you are representing your input specification. So, in this case also, the Verilog and VHDL you write it in some language, then will be represented there are many paradigms to represent them. Ok like trees and graphs. So, in this case we have what do you call control and dataflow graph which is called CDFG, this is Dejong's Hybrid Flow Graphs are there, SSIM flow graphs are there and finite state machine with data and these things are there.

So, the idea is saying that just like our trees and what do you call trees and what do you call this graphs adjacency matrix so many model are there. So, in this case also, when you are representing your hardware language or hardware HDL. So, there are lot of many such models are available like CDFG, then SSIM, finite state machine with data etcetera. So, in most of the widely accepted one is the data control and data flow graphs which will be using in our lecture. Out of these you can see the differences between the general analog handouts which you will be given in the lecture.


So, we will see that. So, CDFGs what we will do with this CDFG? CDFGs what we will do that we will take them .We will take the HDL definition that is Verilog definitions we will take and we will convert them into a control and data flow graph. So, the advantage of this is that this will be very easily useable by your procedures or the programs which will used for conversion of high level synthesis or converting actually the input specification to architectural design. So, this procedure is actually called high level synthesis. So, hence forth in the lecture wherever you say high level synthesis algorithms so you mean that the algorithms take as the input specification in terms of HDL, Verilog or VLDL and it will convert to some architectural design or high level design and the intermediate representations are control and data flow graphs.

(Refer Slide Time: 15:18)



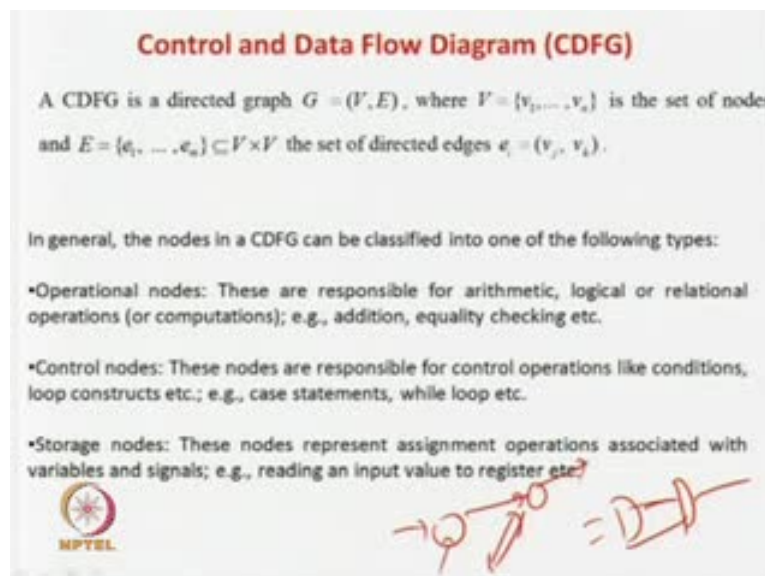
Introduction

- In this lecture, we will discuss CDFG paradigm for modeling of high-level hardware descriptions (given in Verilog).
- CDFG is one of the most widely used modeling paradigm and the others mentioned above are not much different; for details of other paradigms the reader may look into the respective references.



Ok and sometimes I mean what is the use of I mean use control and data flow graph? So, the control and data flow means there will be control part and data flow part as we will see along with that sometimes we will also, have timing. Timing means sometimes we say that this instruction or this operation should be do able or should be done in such an amount of time. Then according to that your hardware will be designed. So, sometimes we have to put with the control and data flow graph also, we put timing.

(Refer Slide Time: 15:34)


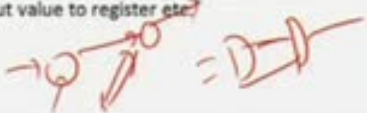


Control and Data Flow Diagram (CDFG)

A CDFG is a directed graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ is the set of nodes and $E = \{e_1, \dots, e_m\} \subset V \times V$ the set of directed edges $e_i = (v_j, v_k)$.

In general, the nodes in a CDFG can be classified into one of the following types:

- Operational nodes: These are responsible for arithmetic, logical or relational operations (or computations); e.g., addition, equality checking etc.
- Control nodes: These nodes are responsible for control operations like conditions, loop constructs etc.; e.g., case statements, while loop etc.
- Storage nodes: These nodes represent assignment operations associated with variables and signals; e.g., reading an input value to register etc.

So, as said that in this lecture we will be using CDFG and Verilog. CDFG is one of the most widely used modelling paradigms. Other things are not that much used I mean people have found various reasons for that. So, that can be found out in the what do you call references. So, let us start with the control and data flow graph. So, what does that? So, control and data flow graph says that it is a directed graph. As I told you as you also, find in our lectures, most of the VLSI problems will be mapped to graphs, because there are very close relationship between graphs and VLSI circuits. Because if you take a very simple circuit, assume the circuit like this very simple circuit.

So, you can always think that in terms of graphs, right? This one graph, so this is another graph nodes and also. These are the inputs and this is the connection between two nodes if there is a path from one graph to another. So, very easily you can see that most of the problems in our digital VLSI design can be very easily mapped to graph. So, in this case also, we will be mainly using the graphs to represent it. So, CDFG is also, some kind of a graph.

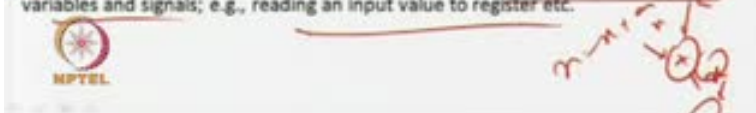
(Refer Slide Time: 16:23)

Control and Data Flow Diagram (CDFG)

A CDFG is a directed graph $G = (V, E)$ where $V = \{v_1, \dots, v_n\}$ is the set of nodes and $E = \{e_1, \dots, e_m\} \subset V \times V$ the set of directed edges $e_i = (v_j, v_k)$.

In general, the nodes in a CDFG can be classified into one of the following types:

- **Operational nodes:** These are responsible for arithmetic, logical or relational operations (or computations); e.g., addition, equality checking etc.
- **Control nodes:** These nodes are responsible for control operations like conditions, loop constructs etc.; e.g., case statements, while loop etc.
- **Storage nodes:** These nodes represent assignment operations associated with variables and signals; e.g., reading an input value to register etc.



So, we are saying that CDFG is a directed graph, where G is equal to V, E that is vertices and edges. So, V are the v_1 and v_2 are the vertices and the set of nodes are the vertices and there are some edges. So, this about the idea. So, it is just like nothing but I will take an example to tell you I mean then there will be more clear that what is the I mean what is the more emphasise I mean it is the directed graph, that what we put exactly more on

this graph. It is not a simple diagram, here we put more information on that, but the structure is very simple to a normal diagram. It has nodes and edges ok.

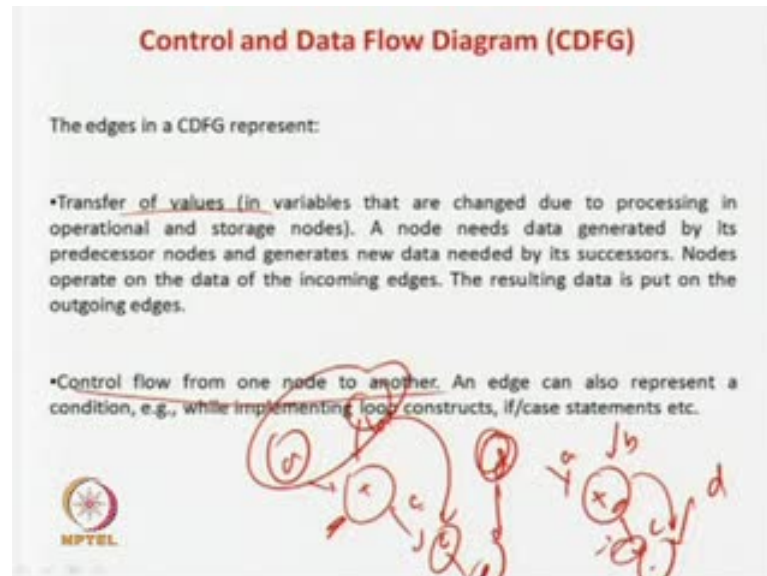
So, now the nodes will be of three different types. So, one is called the operational node. So, what is an operational node? The responsibility of the operational node is for mathematical logic or relational computation like addition and equality checking and all. So, you can say that there these nodes represent some computations. What is CDFG? CDFG will be nothing but will be some graphs. So, we will have some nodes and some edges. And then this is just the digraph. Now these nodes will be of three types. So this is not a simple diagram so we have to put some more stuff on the digraph. So, it is which is making into a control and data flow graph. So, what are the three? First step all the nodes are divided into three classes. First is the operational node. So, what is the operational node? Operational node will do nothing but they are actually something like which will do your mainly computations like addition, subtraction, multiplication, equality checking and all. So, you compute thing into your heart of the computation.

Then another case of node is actually called the control nodes. What is a control node? Like if we sometimes have a statement like if x equal to a then this or this, so there should be some node which will actually (()) it is something which will check the equality. So, equality checking can be done by the operational node that is fine. Based on the answer of the equality checking so some node should be there which will be allowing to take you to this path and there will be another node I mean another set of path depending on the value of x equal to a result, it should be taking this path. So, what is the control node do then? They are responsible for control operations like condition loop, construction case loop etcetera. That is there will be operational node or computational nodes will give you the result that, that is equal to b or something like that is 0 or false or true. Then this control nodes will take or do not take it over from these operational node and then it will decides on the result whether I should go by this path or either I should be going by this path.

So, that is actually the control node then the third types of nodes are nothing but storage node like for example, when we say that x is equal to x plus then there will be some kind of addition will be there. Some x will be there, a will be there and the output again will be generated will be an x . So, there will be some node like say for example, some node which will store the value of a . They are nothing but registers. So, storage nodes are

nothing but they will represent assignment operations associated with signals and variables reading an input variable etcetera.

(Refer Slide Time: 19:10)



So, they just represent that you add something and store the data. You read something and store the data. So, there is actually the use of register variable in most of the hardware definition cases. So, there are three types of the nodes which are in the control and data flow graph. Now let us look at the edges. So, as you already see that control and data flow graph is nothing but it is the digraph. So, the edges are having three types like operational nodes control nodes and storage nodes. Now we will look at the what do you call these? Edges. Now what are the edges do with that? So, edges represent what so that there are two types of edges or represent two things, transfer of values; that is one and the control flow from one node to another.

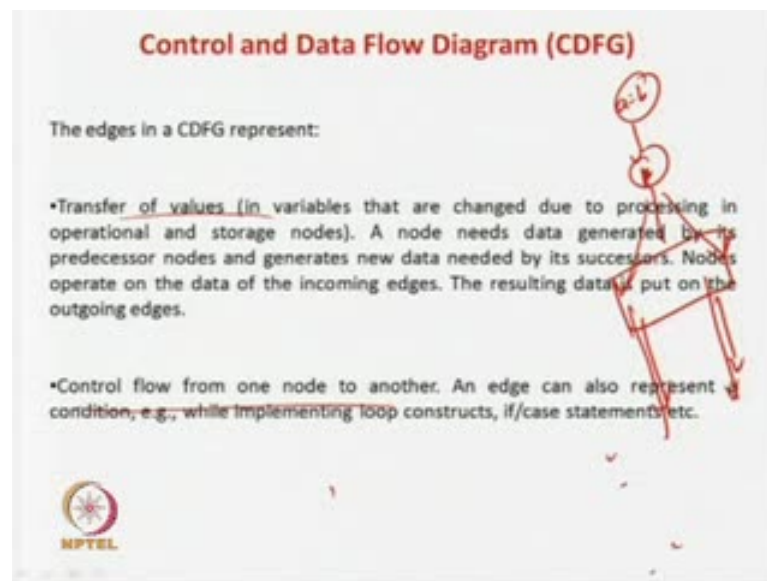
So, now what is the transfer of values like for example, you have plus a, this is a, this is b and it will generate c. It goes to another nodes in minus this is d. So, this edge you can say that it corresponds to a transfer of value. So, what happens, it converts that is it takes a and b, value is added to c and this value of c is taken from this node to this node. So, this edge is nothing but the actually called a transfer of value node.

So, it says that n variables that are changed due to processing elements in the storage nodes. So, I mean transfer of values in variables that are changed due to processing in some operational node. So, here something will be changed and then you have to

obviously there is something storage also there. That you cannot connect like something like this a plus, this is actually say this is a, b and you get the answer as c. So, there is actually some storage node will be there. That is for value of c it will be stored over here and then you can go for this minus operation with d. Right. So, they are also, storage nodes.

So, a and b are in this storage node, in the output will be c. So, c will see taking the value of a plus b and the storage node and then it will be going to the subtraction node and then d will also, there c minus d will be there. So, the transfer of value means what, it is taking the value of a plus b and transferring the value to c. So, from some operational node like this is the operational node, from this operational node it will take the value to a storage node. So, that is nothing but you can see that it takes the values of some computational node and writes it into the storage node. That is the only thing only idea of transfer of value from edges.

(Refer Slide Time: 21:13)

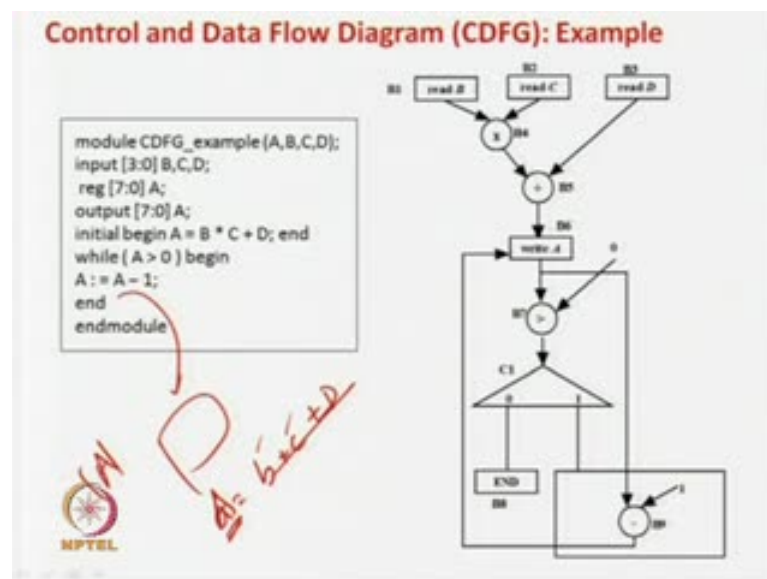


What is the control flow node? So, control flow node is something like say for example, you have an edge like a equal to equal to b something like that. Now you take this value. Now this there is some kind of a let us think something like this. So, this is your a equal to equal to b checker then there is sum, this is the result. Then it is some values here and then after that there is some node. So, a plus b equal to the result some result equal to intermediate result that is e e e a equal to equal to b you get is true, then I think you get

the value as one if it is not equal then you get the value something like this . So, that is the storage value of a result kind of a thing and now what happens if it is true then you will take this path and if it is false then you will take this path.

So, this is nothing like control flow. So, this corresponds to flow of control that is it will decide that whether the control will have to go from here or the control will go from there. We will elaborate all this things in our next few slides elaborate example so the basic idea is that only there will be two types of edges that is very clear. So, one type of edges will represent transfer of data from one node to another mainly it will be from operational node to a storage node kind of a thing. Ok and control node and control flow from one node to another is based on some results of some condition or something it will decide which path to take. Therefore, there are like if then else loops and all those things. So, there are two types of edges.

(Refer Slide Time: 22:27)

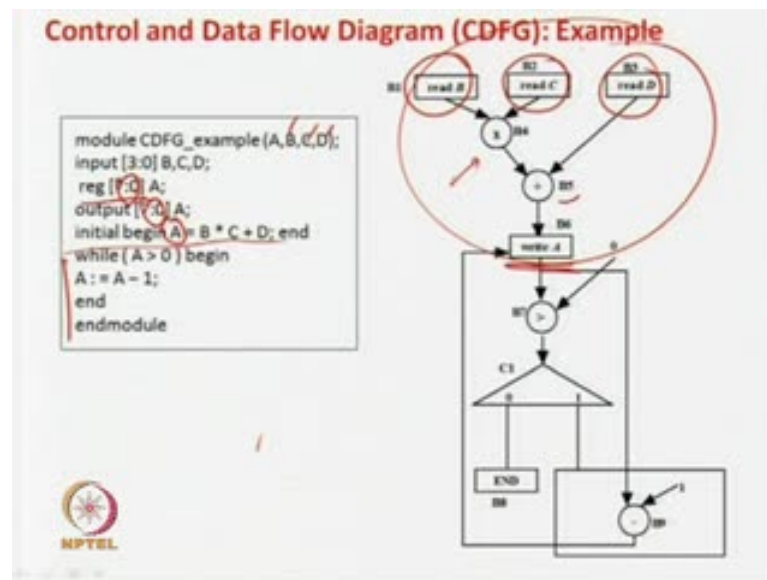


So, I mean what you can say CDFG is nothing but a digraph where the nodes are of three types like operational then computational and so operational and conditional like these three things like operational and computational is same control and storage. And edges are of two types transfer of value and transfer of controls. Now let us take an elaborate example or specific example and see how it is represented. So, whatever idea is there our idea is that something is that so initially we have a equal to b star c plus d. So, this is initial value we have we will read the value of b c d from some outside that is from the

interface and then we have a loop while a is greater than 0, a equal to a minus one that is initially you load a with something, some value and then we execute a loop. Then loop is while a is greater than 0, a equal a minus one we have here there will be some operation like if it is counting the loop there is a loop which is running a num mod of a number of times, then what is the value of a value of a is equal to b star c plus d.

Now you can put some computation which you have not shown here because that is not relevant for this lecture that is, some computation can be there that is that there is a loop that executes b star c plus d number of times and based on that you can do some computations. So, this computation you have not shown because that is not of much relevancy. So, now this is how you write it in Verilog, we say that into module CDFG example this is the name of this one and then you say the a b c d this is one of the inputs and outputs are defined over here. Then we say that the inputs are three four bits 0 to three b c d and a is an output.

(Refer Slide Time: 24:03)



So, we say that output seven 0 this one and also, we say that a 7 0 is already raised that means, it is an output it may store the value of this one and then we say that initial begin, that this initially what going to happen? We say that a equal to b star c plus d and then you have a loop.

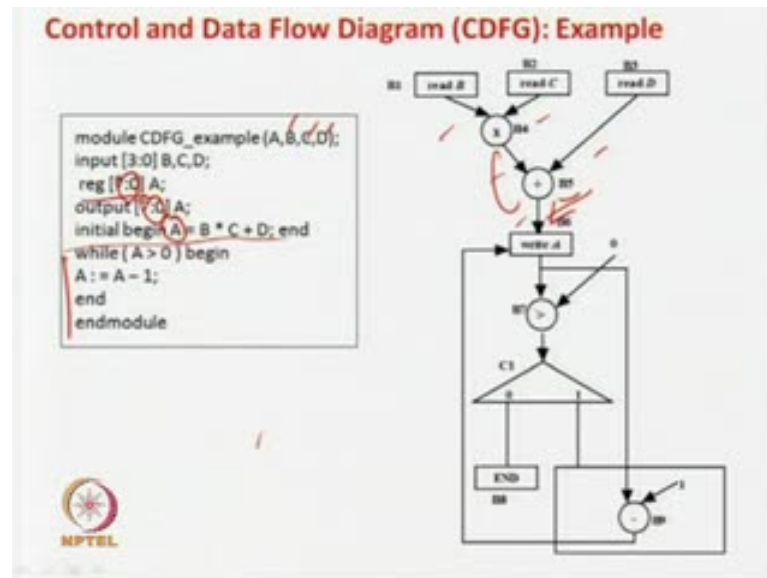
So, that I told it is very simple like C type of language. So, you can just take any reference Verilog reference and can find it, it is very simple and it is just the representation like c with some small differences because obviously c is for software representation and Verilog represent hardware. So, obviously there will be some differences so I mean you can just it corresponds value to VLSI design type of courses so you can just have a look at it and so Verilog codes are in, but it is very simple.

So, just you can see it is very highly correlated with C. So, initially we have b plus c plus b star c. d and there is a loop and actually we have to just declare that the output is of eight bits output we are storing in some registers so defined as a reg and inputs are four bits and so input is take direct kind of a thing . So, these are Verilog code for this one. So, we are not going into very depth into registers, why inputs are not defined as registers and all that so that it represents related to Verilog design, so you can have a look at it in any standard book.

Now how do you represent as a CDFG? So, we have three input variables like b c and d. So, they are read from some external inputs. So, we have three storage nodes in them b, c and d read b read c read d. So, they are nothing but where actually your storage nodes and we number them as b 1, b 2, and b 3. Now what happens you see that we are first doing the star b star c. So, b four is a operational node or computational node which is actually taking the value of b and c and it is writing the value of this one, then we are writing the value of d. So, reading the value of d and it is to be this one. So, here b four and b five is another computational node ,which is adding the value of d with b and c and finally, where it is to be written it has to be written to the value of a. So, this is actually having the storage node.

So, we are adding a equal to b star c. So, this part is literally representing the initial part of the design. So, here we require some kind of a storage blocks and these are the two computational blocks. Obviously you can say that about the edges will tell you later. So, all this things actually this, this, this, this, this and this edges are nothing, but your they are your I mean data transfer edges because in this case is reading the value of b reading the value of c and transferring it.

(Refer Slide Time: 26:04)



So, this will be also, a kind of a data transfer. So, it takes the value of b plus c and it will write into this case. In this case also, it will taking the value of b star c adding to the value of d and writing to the value of a. So, they are all nothing but all they are data transfer based edges. So, formally also, we see in the next slide mainly how, which are and what it is or what. Now you see here this is another computational edge, so it compares a with 0. So, and then it will give you the result. So, if the computation is correct that is a is I mean what you call a is less than 0, so the answer is false. So, you come here and you stop over here. So, this is nothing but a conditional node.


Ok this is a conditional node which is represented by triangle. So, now if a is greater than 0, if the answer is true then what is the answer is 1. This node is nothing but a computational node which is doing a comparison. It is comparing a with 0. Now if you are able to do that so if the answer is true, that a is greater than 0 then it is very good. So, you are giving the answer as 1, then you are going to this computational block again and if it is if I mean a is greater than 0 it becomes false, ok then you will getting the value of 0, it is the CDFG terminates.

(Refer Slide Time: 29:01)

Control and Data Flow Diagram (CDFG): Example

- There are three input variables (B, C and D) which must be read from input lines to registers. So corresponding to reading of each variable in registers we have a storage nodes; B1, B2 and B3 are storage nodes.
- In the Verilog code there is a one time computation "initial begin A = B * C + D; end". For this computation we see that there are 2 sub-computations, namely "*" and "+". So we have two operational nodes, B4 and B5 for "*" and "+", respectively.

The edge (B1, B4) corresponds to transfer of value of B, which gets changed (i.e., new value read) due to processing (reading) in B1. The edge (B4, B5) corresponds to transfer of values (B and C to "B*C"), which get changed due to processing ("*") in B4. After the computation "initial begin A = B * C + D; end", the value is stored in A; this is captured by storage node B6.



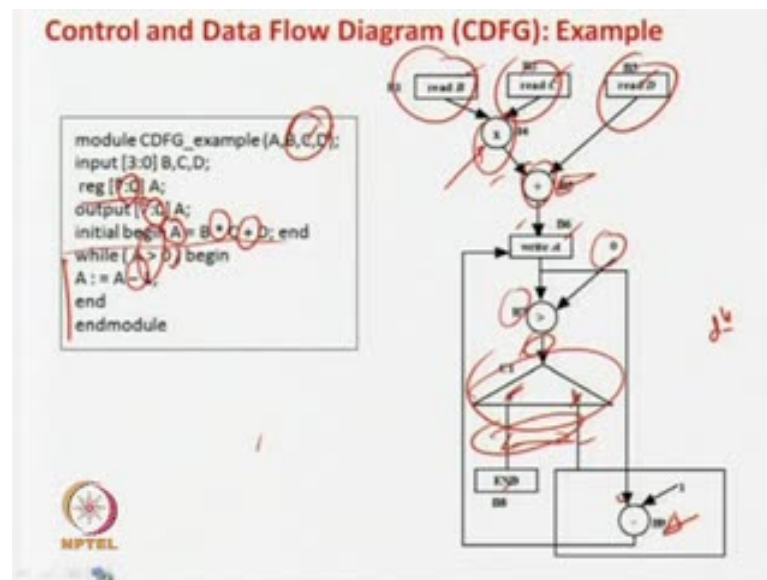
But if it is not the case this is the one which actually come over here and then what happens, if a will be decremented so it will it will read one value of a you see this read over here is decrementing and again writing the value of that to a. So, this is the CDFG representation of this code, is very quite simple. Now I mean these are the storage nodes these are the you can say storage nodes all the rounds are actually computational nodes and you can say this is a control node.

Because, it tells you that when we have to take this path and when we have to take this path. It is quite simple and very efficient representation of a hardware definition language like this one to a graph representation. And about the edges if you see most of the edges will be nothing but your data flow edges, because they are actually all responsible for data flow only. You can think this is two guides which will actually depending on your defining your control. This is also, giving the result they are all giving you the results kind of a thing.

So, right you can see that here we are actually giving with the results. So, what is the result? So, b is greater than this one. So, this result which is actually given that is this component which is giving the result. If the comparison value is 0 to stop comparison value is true, then you come over here that was the discussing and now this is true, but this you can take as the control node as these are the two paths which is residing on the floor.

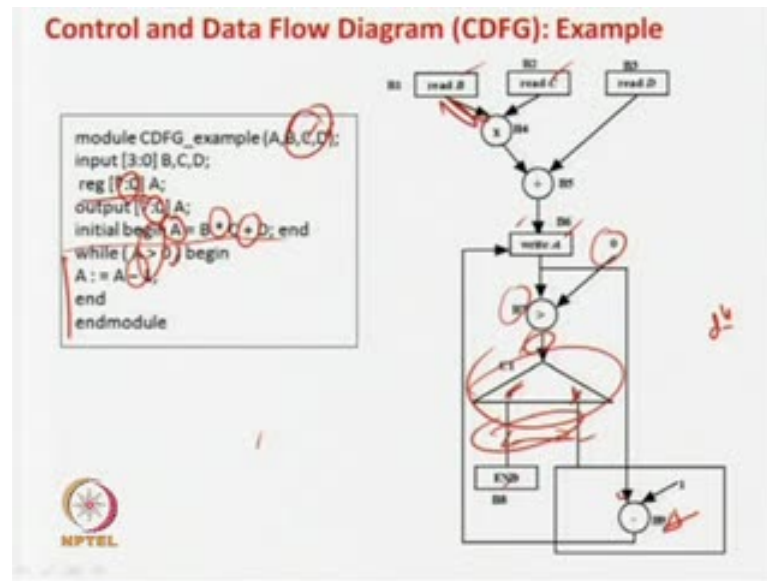
So, I mean mainly these are the two edges you can think they are representing your control and all the edges mainly will be responsible for what do you can say is about your data flow. Now let us see whatever I told let us just put it formally. So, there are three input variables b, c and d which must be read from the some input slides. So, they will you read actually a, b one b two and b three as three storage nodes. So, now whenever you see that Verilog code in any hardware language you see there are some input variables. So, what you can do is that you can straight away say that when we require three registers for that and registers for that or in other words, we require three storage value in diagram what do you call these are storage variables. So, registers or storage nodes also, directly you can map it.

(Refer Slide Time: 29:19)



Now for the initial computation, write this one. So, you have two sub computations star and this one that is true. So, this is for one computation and this is for another computation between this adders. So, we have two operational nodes B 5 and B 4. So, that is what they are saying whenever you have some computations like this one those many numbers of operational nodes or computational nodes you have to put. So, this one star represents B4 and this plus represents addition. So, B5 represents the star. Similarly, for this one also, we will have a operational node like this B9 and this comparison also, we will have another node B10. So, there is whenever you find some operations like greater than equal to plus etcetera you have to do that.

(Refer Slide Time: 29:58)



Ok now what do you do now let us so this is about the what do you call this what means storage about the nodes. So, now if we see the edge see B 1 and B 4 as already told you so there is a node between a is between B 1, B 1 and B4. So, this one edge between a and B 1. It is nothing but a correspond to transfer of value b with guess change I mean you should know you should not change the formal term of change is used then a new value rate. So, if a new value rate are actually also, called change due to processing of reading of B 1. So, what is that why it is a data I mean what do you call rate transferees because the value of b initially was garbage, but now we read a value from the input which is some predefined good value or reasonable value. So, it gets transfer from B to B 4. So, it is getting read from B to B 4 correct.


So, that is not the here. Change means reading value is also,, change like initially registers are some garbage. Now you read something which you get a meaningful value. So, that is also, a change in the variable v. So, if reading is also, a change so this corresponds to a change in value of b so it is the data transfer edge ok. Similarly, edge four five controls to transfer of values this one with gestures. So, all this variable what you saying that is all this edges mainly this are all data transfer based edges. So, let us show this control (()). So, B 7 is the operational node. So, this your B 7. So, this is your operational node again we checks your greater than or equal to staff ok.

(Refer Slide Time: 30:40)

Control and Data Flow Diagram (CDFG): Example

- There are three input variables (B, C and D) which must be read from input lines to registers. So corresponding to reading of each variable in registers we have a storage nodes; B1, B2 and B3 are storage nodes.
- In the Verilog code there is a one time computation "initial begin A = B * C + D; end". For this computation we see that there are 2 sub-computations, namely "*" and "+". So we have two operational nodes, B4 and B5 for "*" and "+", respectively.

The edge (B1, B4) corresponds to transfer of value of B, which gets changed (i.e., new value read) due to processing (reading) in B1. The edge (B4, B5) corresponds to transfer of values (B and C to "B*C"), which get changed due to processing ("*") in B4. After the computation "initial begin A = B * C + D; end", the value is stored in A; this is captured by storage node B6.




(Refer Slide Time: 30:53)

Control and Data Flow Diagram (CDFG): Example

B7 is the operational node that checks 0 with A; output is 0 if $A < 0$ and 1, otherwise. The output of B7 (carried by edge (B7, C1)) controls the control node C1. (B7, C1) is the control flow edge, which corresponds to the condition ($A > 0$) required for execution/exit of the while loop.

Node C1 is a control node responsible for deciding data flow direction after the condition " $A > 0$ " is checked at B7. If value transferred by (B7, C1) is 0 then the loops exits at B8, else computation " $A = A - 1$ " is done at B9 and the loop continues.

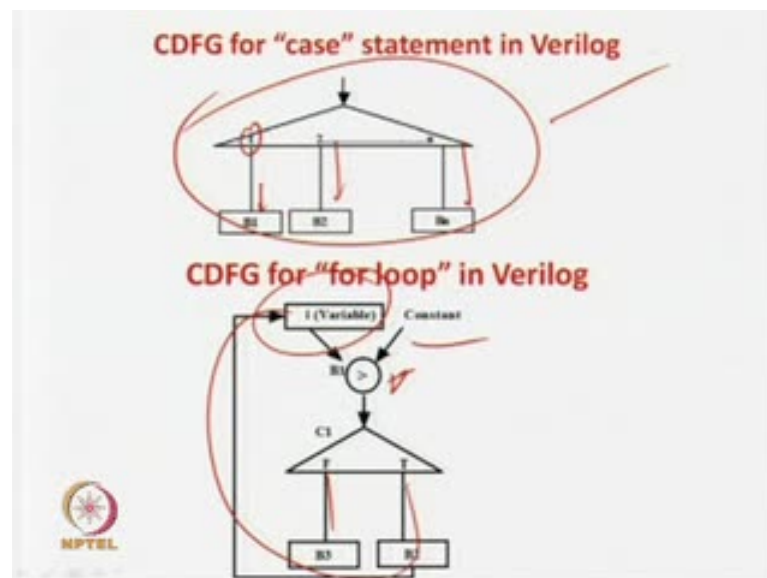


Ok so that is what we will see B 7 is the operational node that checks 0 with a output is 0 if a is less than 0 and one otherwise. Ok the output of b is carried by B 7, C 1. So, the output value that is true and the output value that is this one, so the result is carried by B 7, C 1 that is the C edge. So, the output value B is calculated by this one controls the control node C 1.

Now B 7, C 1 is the control flow edge which corresponds to this one and this one. So, therefore they will say that so this guy this guy and this guy this three parts actually if

you can say they correspond to some kind of a control nature right. So, it will take the value of greater than checking and then this will be corresponding to which part you should take. So, this three nodes exists you can say that they respond to some kind of a control behaviour and all other edges that corresponds to some kind of data behaviour edges, that is what is the saying. Now C 1 is the control which responds to very well for designing the dataflow. So this is checks, so this one and this one etcetera. Whatever I said is it comes under the second point. So, that is what it essentially it says that so all other edges corresponds to data transfers in some form or the other, but three this three edges mainly corresponds to control of some stuff and is mainly the control node.

(Refer Slide Time: 32:13)

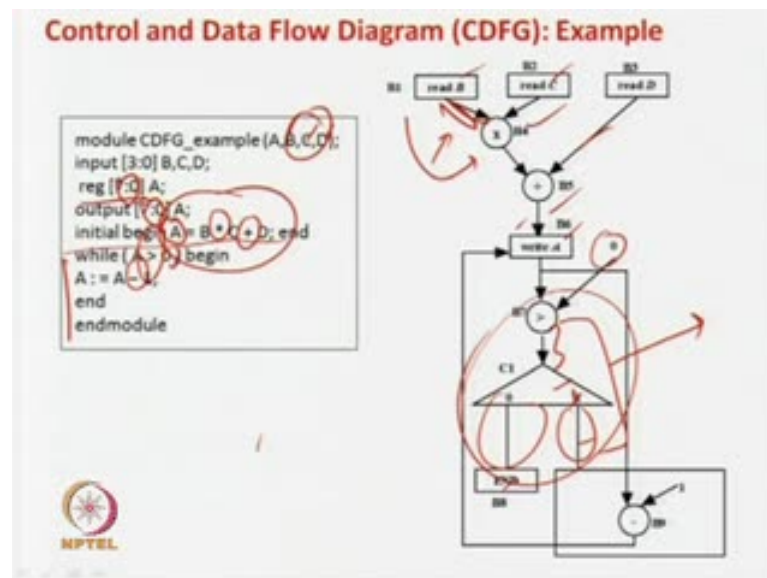


So, this how we can divide our CDFG in to control node I mean control paths and data paths, data agents, data transfer edges or control edges and as well as the nodes into operational nodes, computational nodes the same thing. Then some what you call control nodes and some what we call storage nodes ok. So, this is how even our Verilog code is very simple to convert it into CDFG. So, what you have to do? If you see some variables just make some what do you call from storage nodes for them. So, if you see some operators like plus, minus, multiply, divide, greater than or equal to, less than or equals to. You can go for operational or computational node for that. And whenever you find the sum if then else condition is there, while condition is there etcetera. We have to go for a control node and we have to decide accordingly.

So, if they vary transform from transform from any HDL to a control and data flow graph. That is why data flow graph highly acceptable or accepted in the cad industry. So, other representation like financial statement data and ssim models and all we can discuss when you read through the literature I mean the references given in the hand outs allowing this. We will find that they were much more complex and they was not easy to transform from one form that is HDL to those things, that we can find out easily ,that is why CDFG is well acceptable.

So, like in this case if you have case statement in Verilog it is like for case one we have to go to B1 and for case two we have to go for B4 and for case n we have to go foe B n. So, this is a way of simple representation of CDFG for case statement for loop. It is also very simple like they will be a variable i so you will have a storage node for that and it will have a computational or computational node and have the operation node that will compare the value if some constant if it is true, it will go for this loop or it will stop there. So as we said it is very easy one to one representation, we did one form of representation to another.

(Refer Slide Time: 34:05)



Now I like to told you that timing is very important in Verilog design or in any hardware design. Like, for example, sometime we say that two like in this case of Verilog code only we say that we do initially a is equals to b plus this stars. Now there is a very important point over here is what specifies any time over here. So, what is time over


here? Time means if you say that this operation has to be done in say has 25 nanoseconds something like that we may say. Then what it means? It means that because of some input parameters or some input constraints so we will elaborate this things in the sequence of (()) they going to have that why timing is so important.

So, it says that something is taken in some lament language you can understand this. Say, for example, there is a chip ok. We says that we will give some inputs and say that this clock pulse, but this clock pulse say give some inputs sequel to one or some dot dot like that and that the end of say two other clock pulses you should get the answer over here and if you take some sequence some frequency over here . So, within one by f we have to get the value. So, then within this time period say 29 nanosecond. So, we will say 25 by 2, 12.5 nanoseconds is the time period and we say that in two clock pulse that is 25 nanosecond you should get the answer. That means what you have to put some time that you cannot vary indefinitely. That you give a result then your answer will coming out another say in 10 days, it will vary slowly after operation that is non-tolerable.


So, for some operation because whatever we are paying suppose he may give the input for the compute, you will get the answer in two clock pulses or 29 nanoseconds. Some other part of the input he may give it to some other part of the input they will arrive here. Finally, use this to staff and you have to generate the output. Now the problem what will happen is that if you are what you can say that? If you have very well designed circuit, you can answer that preferably in 10 nanoseconds, but then still this answer has to be arrived over here. So, that the both of they coming to there must be conversed so added to the results. So, I mean this will become the bonding. If you think that instead of 25 nanoseconds it is taking twenty five mille seconds, then you are under a big problem because even if this block is very fast, but because of this slow operation of the block nothing works actually. OK. So, that is why timing is the very important timing concept or timing is very important paradigm included in.

(Refer Slide Time: 36:06)

CDFG with timing



- Sometimes minimum required delay needs to be mentioned in the specifications.
- Delay information is required in logic synthesis. Delay of a circuit depends on gates and architecture.
- For example, speed of ripple-carry adder is lower compared to carry-look-ahead adder; however, area of carry-look-ahead adder is higher than ripple-carry adder.
- Also, faster gates consume more area and power compared to slower gates. So, if tolerable delay of some operation in a circuit is low (which depends on application), realizing it with faster circuitry unnecessarily leads to high area and power overheads.
- In CDFG, delay information is modeled using a node between the required points, where the delay is specified. The points can be between single operational nodes, between multiple operational nodes, loop etc.



Sometime other way around also, you may design a block like we have done very hard work and says that there are two blocks over here. So inputs are coming to this and finally you join another block then you get the new device. If you also, work very hard on this you have taken some 10 nanoseconds to generate this output ok and for this is also, done by very intelligent people and they have also, work very hard, but with lot of effects also, they could not generate within without less than 25 nanoseconds. That is whatever you do, the best resources is we have unseen everything, but we complexity of the computation was so high. We could not generate any result with less than 25 nanoseconds. Now what we will do? So, deliberately what you can visit? You can increase your result also to 25 nanoseconds because unnecessarily no point in getting the result here anyway you have to wait for this.

Now what is the thing? Coming in design what happens? If you are going for a very fast design so you will also see the elaboration in the sequence of lectures modules. So, if you take a very, very fast design, very fast adder, you know a ripple carry instead of slow adder there is look adder and carries a adder, carries n adder etc. and in carry look adder you can much more gates than the ripple carry adder.

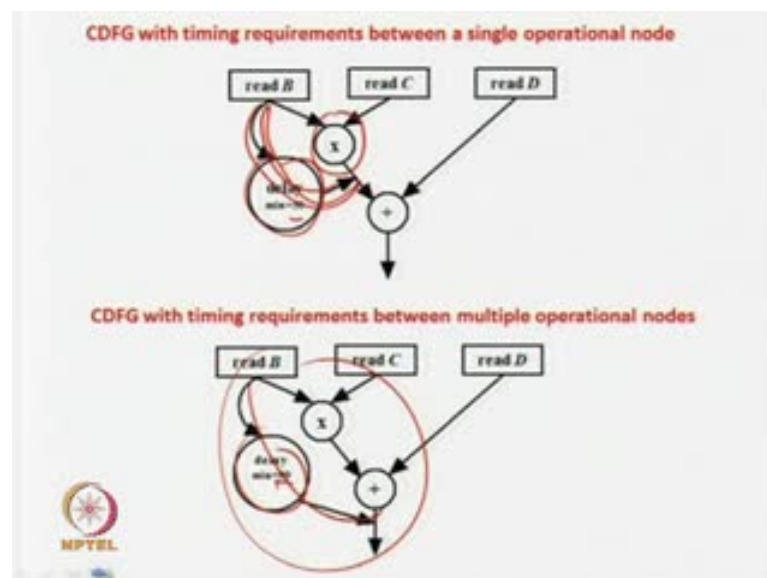
So, in the thermally negative ready for taking fast circuit, or if you are taking a very high end I mean the computations are very fast with very fast generally the power consumption and area of those layers are larger. We, so now in this case anywhere the

fastest possible gates and all nothing less than 25 nanoseconds. So, why unnecessarily you want to make it 10 nanoseconds and use more resources when it is of not any use because to use the result which arises here, so they should wait for 29 nanoseconds. Here we have given the results in 10 nanoseconds fine, but you cannot do the other results, you have to wait for 29 nanoseconds.

So, in this case what the designer will do? Actually we have to relax our design requirements. So, that he gets he also, designs it in 25 nanoseconds solution and the idea is that that means both the design will arrive in 29 nanoseconds and your job is done and from 10 nanosecond if you going to 25 nanoseconds timing. So, your area of opponent will be less as well as your power requirement will be less. So, that is why timing is very, very crucial that it is not only for that means setting the deadline and also, for setting the delay. Both of them you can set the timing factor. So, that is why I am saying that timing is also, very important parameter in input specification so our CDFG should also, including the timing paradigm.

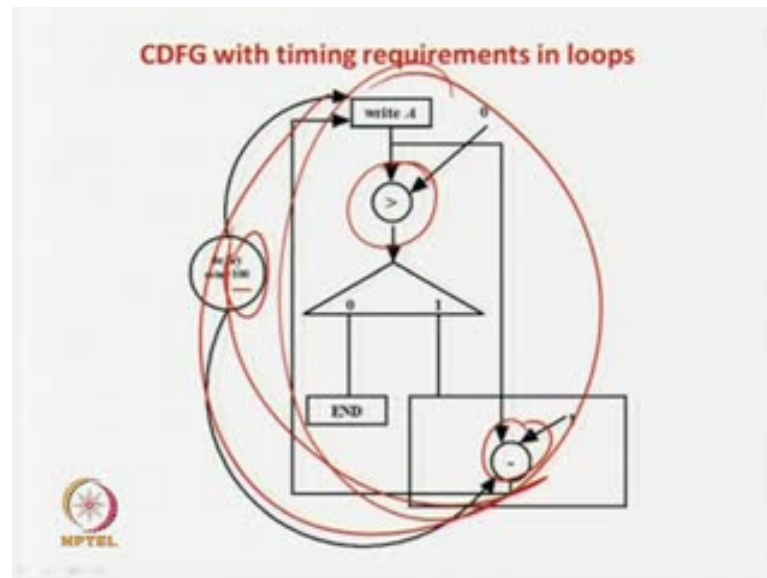
So, like that it is what we said so minimum time required to be mentioned. Sometimes minimum required delay needs to be mentioned in the specifications. So, like this ripple carry adder is a slow adder so corresponding this one. So, if there is very, very fast circuit more decay will be so on looking very carefully when we have to use a fast circuit and when we have to use a slow circuit.

(Refer Slide Time: 38:32)



So, we will see how in CDFG we can know that. It is very simple in CDFG say like for example you want to do CDFG the delay of this one is say, 50 nanoseconds. So, you have to put a circle and you have to say that this delay is nothing but 25 nanoseconds. Sometimes you can say that I want that whole combination delay to be 20 nanoseconds, multiple nodes.

(Refer Slide Time: 38:55)



So, you have to put one that means a age with a circular node with this say that it is a 20 nanoseconds. So this is a very simple formal representation or this one that is why it is so widely accepted as a this thing. Now in this case also, you can say that this loop this whole loop so iteration of 1 loop should be able to do in minimum 100 nanoseconds. So, in this case, that will decide the delay of n is equals to there is a subtraction it will decide this comparator block will decide the delay of all those things. So, in this case also, you can see have done this and you have send that the delay between this whole loop is nothing but 100 nanoseconds.

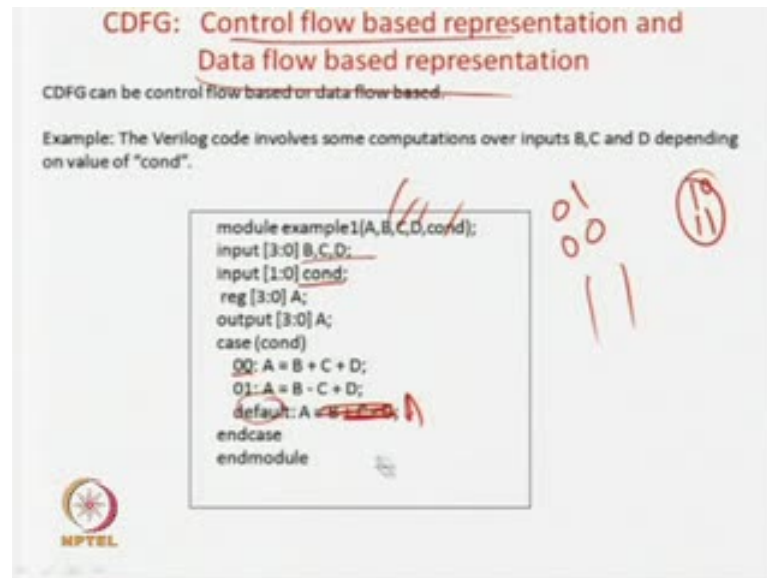
(Refer Slide Time: 39:19)

CDFG: Control flow based representation and Data flow based representation

CDFG can be control flow based or data flow based.

Example: The Verilog code involves some computations over inputs B,C and D depending on value of "cond".

```
module example1(A,B,C,D,cond);
input [3:0] B,C,D;
input [1:0] cond;
reg [3:0] A;
output [3:0] A;
case (cond)
00: A = B + C + D;
01: A = B - C + D;
default: A = B + C + D;
endcase
endmodule
```



So, the more the more what I want to emphasize is that so in case of Verilog design I mean in case of any hardware definition language, important parameters are storage that is input variables some operation and some controls. So, they can be very easily map to the CDFG operation with low storage nodes control nodes. So, we look really for the edges.

Now and there is other part missing is the timing. So, timing should also generally and all other design timing are specified like nobody will purchase Pentium chip series if there is Pentium 1 or Pentium 4. We always say that it is a Pentium core and Pentium dual core, 2 Gigahertz or 1 Gigahertz. So, that is actually your timing. So timing is the very most important paradigm which actually that the this mobile is running in a high android two in such a high memories, processor is 1 Gigahertz the processor and they are all actually marketing terms as well as they will decide the performance of yours devices or circuits.

So, as they tell you has this is the performance of the circuit and so forth hence therefore, they are very, very important paradigm. Ok. So, timing will have to be should use all this specifications. Timing are present in all the specification which we take ok. So, therefore timing also, should be represented in CDFG because they have seen that CDFG is a very natural bonding paradigm for your hardware definition languages.

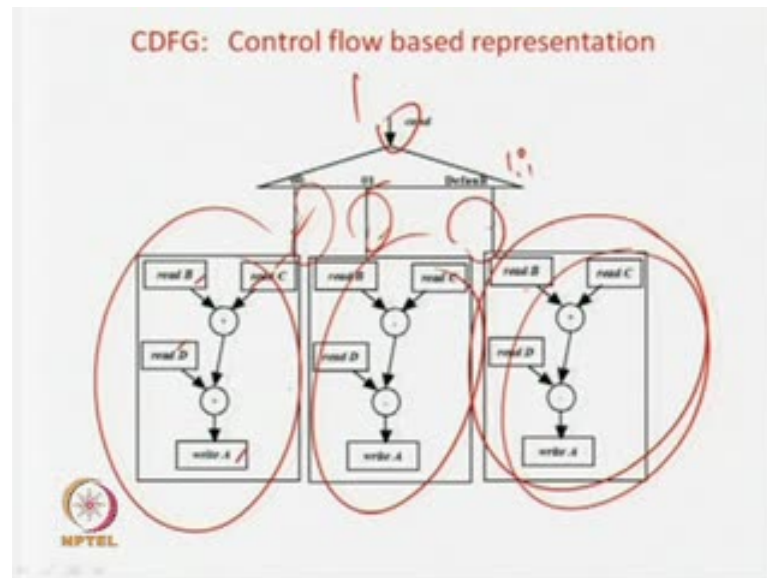
So, they have also, we have seen that timing can be very easily incorporated between the loops, between multiples nodes just like putting an ages saying that this operation must be completed in this much time. So, CDFG control data flow graph and timing is also, very simple representation. So, we may not need an action CDFG have one to one mapping you can say directly to the Verilog. Ok. Somewhat like that. Now we will see they are basically two types of CDFGs and that is very very important called the control flow based CDFG and another one is called the data flow based representation.

Ok thus we have modified our Verilog code, some Verilog code we have taken, then modified in a bit. So, you can see in this case we are taking a, b, c, d and some condition is there, inputs are b, c, d and also, a condition is there it is also, coming for inputs. So, this are all your input block b, c, d and conditions like b, c, d are four bits and condition is a two bit control and output is a in this case four bit output and we say that if the condition is 0, 0; then you write a is equal to b plus c plus d and if you say the control is 0, 1, then you have to write a equal to b minus c plus d and default.

Actually one difference in hardware definition language compared to where C in C language you can say that condition is 0, 0, 0, 1 then you have in case of hardware you have two lines, so conditions you have is about 0 0 0 1 and nobody can prevent from happening ones and 0 and 1, 1, 2 may come they may also come. Nobody can prevent them from happening for some in c language you can made about two things like if there is no else the two ways can say that two conditions only that will happen and very much sure about it and you need not find out any default, but in case of hardware you have to tell because there is no hardware from mean which you can prevent the happening 1, 0 or 1, 1 and you have to add a default case some indifferences of Verilog. So, if we have to add a default case, in all other cases, you have to have a equal to b plus c minus d and also you could have said that something like a is equals to a like that. So, if we give up the default case of hardware languages.

So, do not mix up these things with the current lectures because they are all interims of Verilog languages, this was just offline information for you, but just the idea here is that so depending on the conditions they have given three types of compositions. So, now we will map them to CDFG. So, there we will see that they are two types of CDFGs, control flow and data flow. So, we will watch which the advantage of that is and so forth.

(Refer Slide Time: 42:45)



So, control flow it is very simple. So, how do you do the control flow? So, first you know that b, c, d and the condition. So, these are all input variables. So, obviously you have to have some storage devices. So you see, in this cases for timing you just do not see about this. So you just see it is b c and d. So, they are about your storage registers or they are actually your storage loads and you write the value of this. So, it is storage load and also, there is a condition is there.

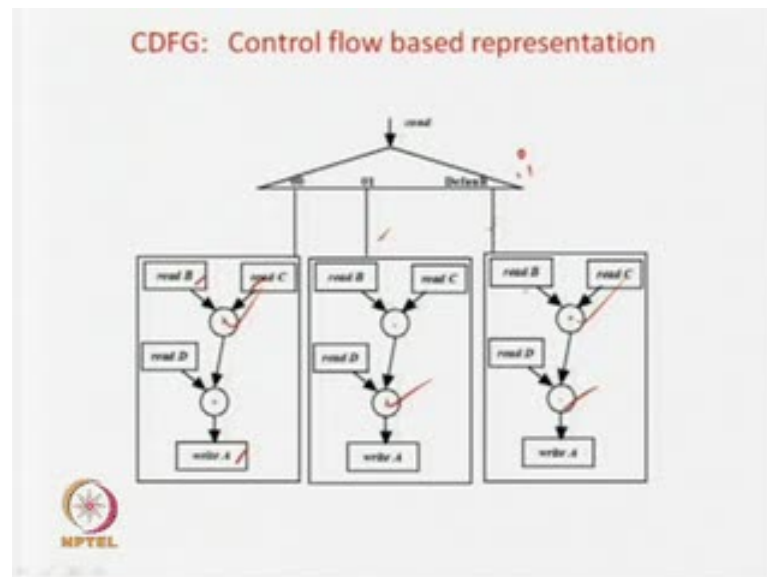
So, you should also have condition variables. So, it is a condition is your storage register right. So, this is storage load then you require two computational nodes or operational nodes because the two some subtractor and so forth. So, mean 1, 2, 3, 4, 5, and 6. So, six these things will require. So, you just see 1, 2, 3, 4, 5, 6 and so some six computational nodes are there. Correct. Now we will see why we replicated something and also, that is integrated details that will see here ok and then what you require for the condition you require a control block. So, this is the control block for this one and that is it.

So, what we have done? So, it is the control flow based representation. So, what we do in this case we see that there are three con I mean what you call there three basic operations this, this and this. So and based on this control there will be three flows. What we do? So, we represent this with one flow this with one flow and this with one flow. Ok. Now

what is the flow if the condition is 0, 0 then a equal to b plus c plus d. So, what we have, we have three storage nodes and two operational nodes.

So, actually four storage nodes and two operational nodes if the condition is 0, 1, then what you have to do? You have to do b plus c, b minus c and minus d you have to do. Ok. So, conversely we meet another block and if the error conditions like one, 0 and one, one, and then you have the third block where you have d d plus c minus d. So, that is what similarly, you have done and now based on the based on the value condition either this one or this one or this one. So, these are the particular what do you call the three data control transfer area, control based stages. So, we have decide which block to take.

(Refer Slide Time: 45:01)



So, this actually called the control based representation of your CDFG for that Verilog code. Now here you can see, the lot of redundancies and all like because you see in this case, b, c, b, c, b, c. So, unnecessarily you have put 1, 2, 3, 4, 5, 6 too many nodes and now you put a read b once, c once because they have to be read only once and depending on that you could have done something.


Similarly for d also, you have replicated it thrice like this one, this operation and this operation nodes are replicated. Similarly, you can say this operation node and these operational nodes are replicated. So, lot of replications if you are going to for the control based representation. Now what is this control based representation? It is nothing but the simple thing you can just see the Verilog code, we have these are, one computation, this

is two computation, this is three computation and you have made three blocks for this and this are condition block.

(Refer Slide Time: 45:42)

CDFG: Control flow based representation

- It may be noted that for each value of "cond" there is a sub-sequence of operations represented by storage and operational nodes.
- Here, the operations are classified based on three values of "cond": 00, 01, default (10,11) and the corresponding sub-sequence of operations involving storage and operational nodes are enclosed by three squares.
- Therefore, the control flow based CDFG has almost a one to one mapping with the lines of Verilog code. So, control flow based CDFG gives an idea that, depending on value of condition of a control node ("cond" in this example) only one sub-set of operators and storages are executed.
- It may be noted that this is software concept because in a program only those parts of a code are executed which satisfy conditional statements like "if-then-else", "case" etc.




(Refer Slide Time: 45:56)

CDFG: Control flow based representation

However, in hardware there is no concept of execution of a "sub-set" of operators/storages (I.e., statements of an HDL code), based on value of condition of a control node.

In the example, variable "A" can be assigned three different values through three different computations depending on value of "cond".

- So, three hardware circuitry are to be kept in the chip and depending on the value of "cond", the output of appropriate circuitry would write "A".
- In other words, unlike a software code, where parts of a code can be invoked based on values of a condition, in hardware, all the different types of circuitry are to be implemented in the chip (and would be executed) and the output being used depends on the value of a condition.

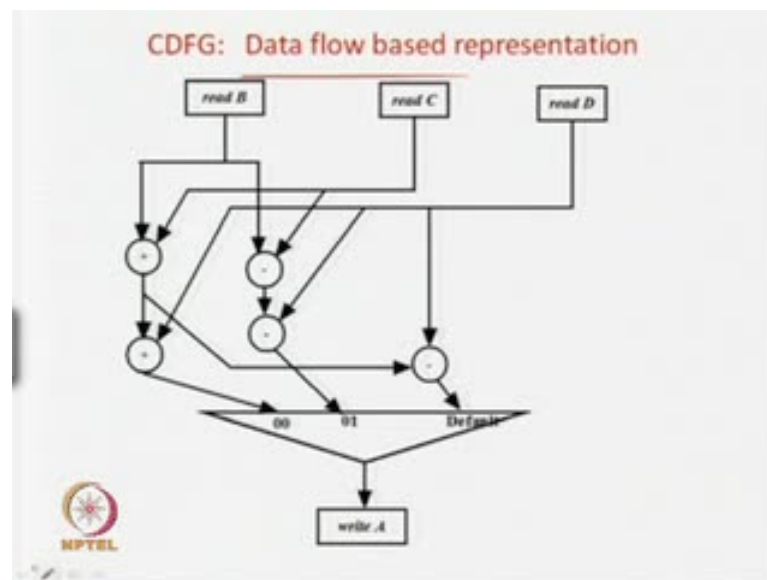


This is the conditional block for that we have to condition node that we have added. So, we have done know one to one mapping from the code to this one. So, this is actually called a control flow based representation. So, it is nothing disadvantage of it the only problem we found out here is that this a, b the advantage of this one is this is diagonal nothing but the code user very easy to do. But there is some kind of redundancies over

there. Now let us see so that this is all the discussion that I have told you that how we are made this blocks. So, you can just have a look at it. So, it is a question conditions, we have made three blocks and all those stuff this is written over in the flex.

So, whatever I told you this is in the slide now those are nothing, but the control based implementation that was been discussed. So, essentially what this slides are saying that in case of control based representation you take a control node, based on each path is like control value like 0, 0, 0, 1, 1, 0, 1, 1, you have four sub modules or three sub modules and you attach them based on the based on the computationally code like. So, there are there are three sub modules you can say these are computations kind of. So, you have three this kind of a stuff, but that is a very simple representation and you can do this.

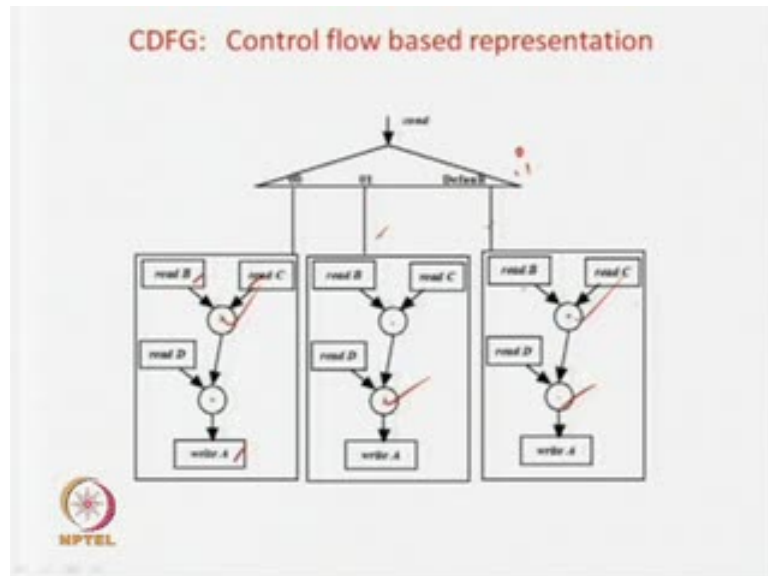
(Refer Slide Time: 46:24)



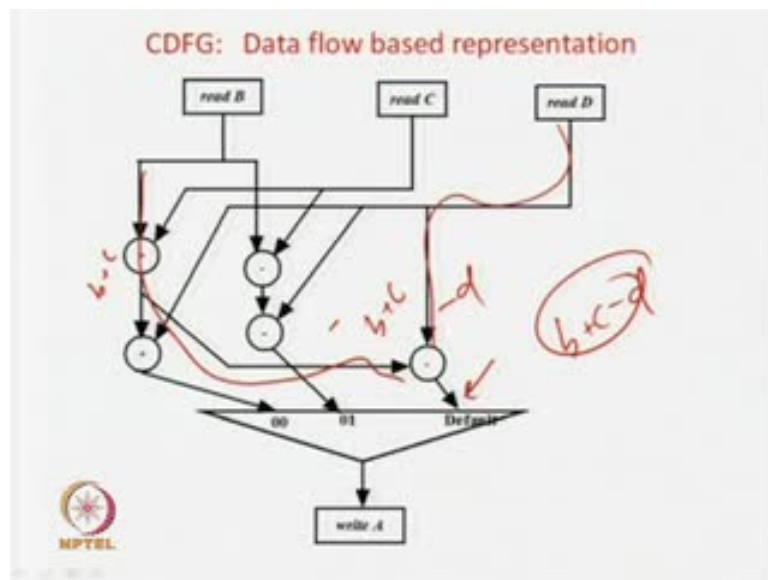
Now how to eliminate the redundancy? Because you saw that we are unnecessarily reading a three times, unnecessarily reading b three times, and this kind of a thing. Ok. So, how are writing a three times and so forth. Though here will see, how we can eliminate the redundancies and we can do that. So, this is actually called the data flow based representation. So, the previous was actually nothing but control flow based representation. Like this control and its sub module. The control that is actually the controller is taking the main prominence here. So, what are the controller, the controller was actually having 0, 0, one, 0 and based on that it was trying to invoke three different

modules. So, that is actually called control based end because the controller is taking your main advantage. Ok. That is what written in this slide.

(Refer Slide Time: 46:44)



(Refer Slide Time: 47:03)

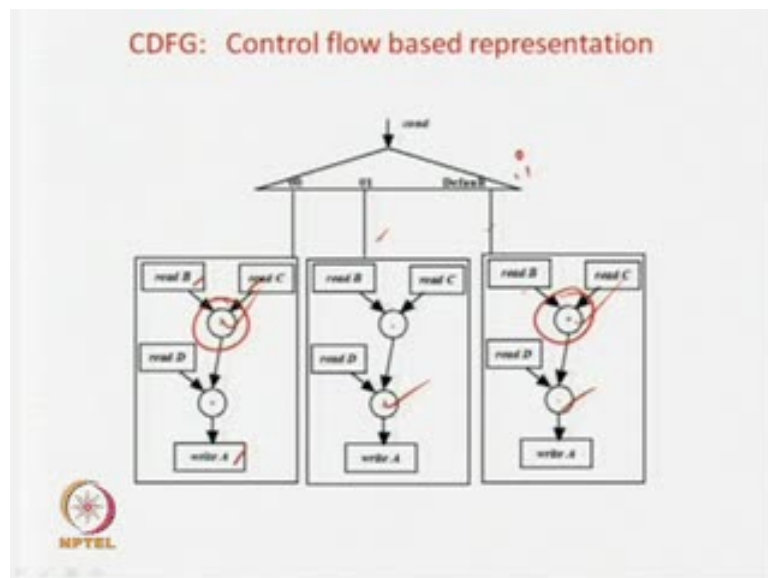


Like what is that, how do we reduce the data redundancy? It also very simple. So, in this case, what we have to do? We have to go data flow based representation or CDFG. So, in this case we not find directly the one to one mapping in the from the Verilog code. see in this case what happens, you have to just go for a re structuring kind of a thing, but here all the what you call redundancies will be eliminated.

So, here you see you read b, c and d earlier once. That is very important now you see what their? The data path or data is taking the more dominance in the controller. So, controller is put at the end. So, it is having 0, 0, 0, 1 and default. So, based on that you will write the value of it that is true, but will reduce all the redundancies we have to see because we will be compressing the data path. So, what is this b, c and d? That is there now you take this path. So, it is saying that b, d this is actually coming from c. So, this is b plus c and this one if you see it is coming from d. It is b plus c plus d . So, this path actually called also, b plus c plus d, but node what you call operational node has been replicated. That is b plus c plus d are keeping the value of this one is ready over here. Now take this second value.

So, second value computation required was what, b minus c plus d, but second value computation required was b minus c plus d what you do? So, this is b minus c minus d. So, this flow is b minus c minus d. So, this value is again kept it over here. Now what is the third value of computation? The third value of computation was I think. This one b minus c plus d. So, this one is what? So, this one is nothing but b plus c this one is again, b plus c and this one is subtracted with d this one is path. So, this is b plus c, this path is coming over here, this is b plus c and then one is subtracted minus d, so b plus c minus d. So, this value is kept ready over here.

(Refer Slide Time: 48:57)

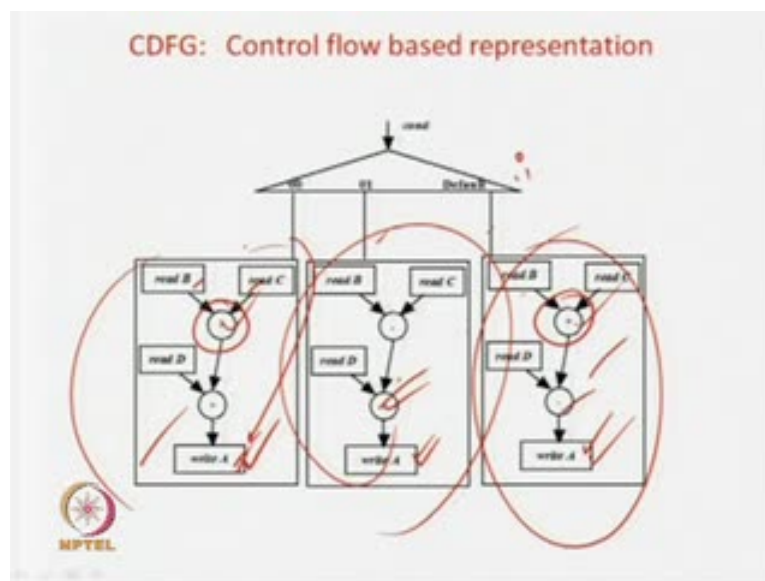


Now you see what happens? If you go for a control based thing, then there is this one and this one is redundant. So, you are actually using two adders or two what you can say that operating, operational nodes here that replicated but in this case we are not doing it. This $b + c$ is used for this case is also, replicated the same node is used for this default case. So, then we are not replicating anything. So, this same stuff is used over here and the result is brought over here.

Now all things actually all this results for all the computations for what you are doing, we are keeping the value and ready here, here and here. Now depending on your control value you will use this one to write over here or this one to write over here or this one. So, if either one is represent for what we have done, we have used the data path of the dominating path. So, like this one data path, like data path this is one data path and this is one data path. All the data paths are operating in parallel and all the results are kept ready and this is the control you will use either one of them.

So, here actually we are able to reduce the redundancy actually we are able to take the redundancy the data nodes, storage nodes, all the computational nodes, every node of the data we have reduced and here got a smaller graph in terms of CDFG data flow based representation. But what will be the disadvantage of this?

(Refer Slide Time: 50:18)



The disadvantage of this is we see that there is no direct mapping from the Verilog code. We have to work manually find out which are the redundant nodes, the redundant

computations and then how can you merge them and to get a most optimal design. So, that is not a one to one mapping you have to again put some algorithm to do that.

And you can say that this case is also , one more case is over there that is very simple in this case of this control flow because there is a one to one mapping from your Verilog code and you can just say that this is one computation, this is one computation and this is one based on that this you either invoke this or invoke this or invoke this, but one thing I have to tell you is that in case of hardware there is nothing called as invoking this, invoking this or invoking this.

It is not software, is not like you call this procedural, you call this procedural, you call this procedural. All the hardware are already available. You cannot you just switch it on and switch it off. Just rest I mean you can say enable we write this one or enable you write this one or enable you write this one based on this value. The all the computations will be done in parallel and you have to enable like this or this or this in the end based on the value of control.

(Refer Slide Time: 51:59)

CDFG: Data flow based representation

The control node is after the operational and storage nodes.

- There are three circuits for computing different values of "A", depending on "cond".
- The three circuits evaluate three different outputs irrespective of the value of "cond"; "A" is written by the output of the appropriate circuit depending on value of "cond".

It may also be noted that even if three different circuitry are required to compute different possible values of "A", many operational and storage nodes are common among these circuits and redundancy can be eliminated.

For example, reading of B,C,D are required by all the three circuits and they may be implemented by storage nodes common to all the three circuits. Also, operational node for "B+C" is common to circuit for "A = B + C + D" and circuit for "A = B + C - D", which can be merged.

NPTEL

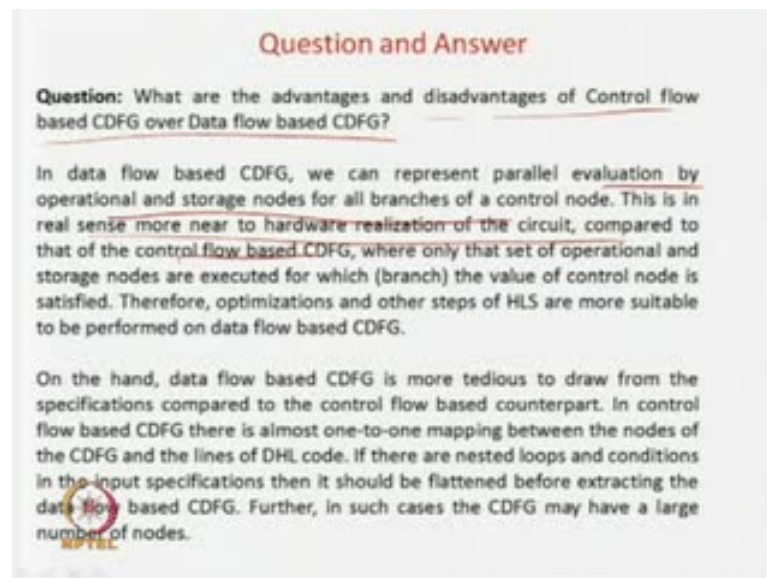
So, there is already this three hardware executing in parallel is because I am in hardware nothing called the this is kept from initial and you bring it down and collecting it. When you are doing your operation not possible because everything is on chip everything is operating in parallel. So, this one and this one and this one everything is operating in parallel based on your condition you can enable right of this one or enable right of this

one or enable right of this one. So, unnecessarily you are putting three redundant hardware's and you are using any one of them based on the requirement. But in this case we have actually.

In this case what we have done? We have reduced the redundancy manually and by looking at the code and doing some other operation and based on this default way of writing. So, this is the basic difference between control based CDFG and data based CDFG. But we should always see that the data flow CDFG is very good always use it because the reason is that it is not directly one to one mapping is not there from the Verilog code to is the data flow base representation.

We have to find out these are redundancy, the how we eliminate this redundancy and so forth. But it is the minimum representation that you can say the number of redundancies are removed and all this mod near to this hardware kind of a thing. Ok. So, this is about the explanation about this control flow based representation which we have discussed so that is what it is written in this slide. So, you can go through them.

(Refer Slide Time: 52:13)



Question and Answer

Question: What are the advantages and disadvantages of Control flow based CDFG over Data flow based CDFG?

In data flow based CDFG, we can represent parallel evaluation by operational and storage nodes for all branches of a control node. This is in real sense more near to hardware realization of the circuit, compared to that of the control flow based CDFG, where only that set of operational and storage nodes are executed for which (branch) the value of control node is satisfied. Therefore, optimizations and other steps of HLS are more suitable to be performed on data flow based CDFG.

On the hand, data flow based CDFG is more tedious to draw from the specifications compared to the control flow based counterpart. In control flow based CDFG there is almost one-to-one mapping between the nodes of the CDFG and the lines of DHL code. If there are nested loops and conditions in the input specifications then it should be flattened before extracting the data flow based CDFG. Further, in such cases the CDFG may have a large number of nodes.

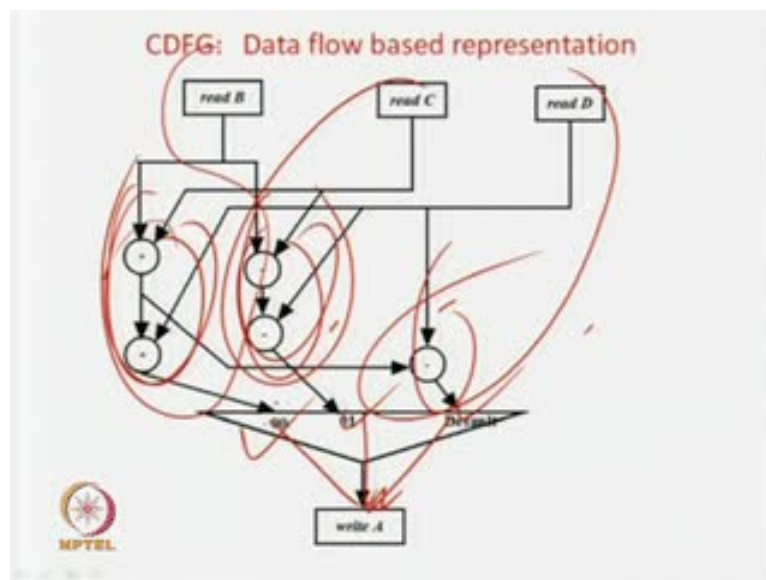
Now let us end this before stopping this lecture revise on the question answer based session. So, what are the advantages and disadvantages of control flow based CDFG over the data flow CDFG. This is just the question we were seen just now. So, as you know that data flow based CDFG can represent parallel evaluation of the operational and

storage nodes for all branches. This is in real sense more near to hardware realization of the circuit controlled to controlled for this CDFG. So, what is the idea?

Like this is how the representation. Like what is we saying is that this is a software representation. If this then this is the procedural, if this then this is the procedural and if this then this is the procedural. It is not very near to hardware, but still you can represent it in hardware by the by the round of with what I tell compute all and enable the writing of any one of these variables which based on the condition. That is not very good because unnecessarily you are going for three different parallel similar type of computation.

So, this is actually called a more like a software representation that is why it is it is very near to the Verilog code. So, whatever you write in the code? You can directly represent as a control flow based CDFG very because one to one mapping it results like a software either based on condition this procedure or this procedural or this procedural. So, that is not a very good thing, but because it is more looks a software that is why it is a very one to one mapping to hardware.

(Refer Slide Time: 53:20)



One to one mapping to your Verilog code, that is what the idea, but in hardware actually we never do like this, we never based on this if then else, we never put three different types of hardware and select any one of them. It is not software. So, what basically we do is that, we compute all the components that is do with the parallel computation, but

we will use minimal number of redundancies because we know that we cannot switch one module of an one module. All this are very difficult between hardware.

So, we compute here also, all this stuff, all the values we compute for the values of a but we will be using the you will eliminate the redundancy and we will select the output based on economy shell. That is obviously they are sending data flow as well as they are sending control also, and in data flow what we are doing? We are also, doing all the computations, but we are not taking these things directly from the code. What you are doing is that we are taking the code, we are merging the operation, merging the common operations and all, and then we are showing that which is the path to be taken and then based on the paths selection will also be done by the control. Path selection like this one, this one or this one or a similar to this module selection based on this control. That is path is same or what we have done?

We have not explicitly copied the modules or the operations of yours Verilog code in three different modules rather what we have done? Rather we have merged this common sub expressions or common expressions and we are eliminating the redundancies. That are actually solve the redundancy problem and that is more near to a hardware relation because if you want to realize in hardware everybody will do it like that. Instead we do not much think about eliminating the common sub expression and all, because in C program it is nothing but two or three lines of code.

So if you and if you also, invoking everything parallely. Based on your condition you invoke either procedural a or procedural b or procedural c. But this hardware, all the three procedurals are invoked at one time and all this three hardware's are executed at one time. So that is why we think about eliminating the redundancy, which is possible in data flow based class. So, these are all the advantages of your data flow base representation now what is your disadvantage?

(Refer Slide Time: 55:48)

Question and Answer

Question: What are the advantages and disadvantages of Control flow based CDFG over Data flow based CDFG?

In data flow based CDFG, we can represent parallel evaluation by operational and storage nodes for all branches of a control node. This is in real sense more near to hardware realization of the circuit, compared to that of the control flow based CDFG, where only that set of operational and storage nodes are executed for which (branch) the value of control node is satisfied. Therefore, optimizations and other steps of HLS are more suitable to be performed on data flow based CDFG.

On the hand, data flow based CDFG is more tedious to draw from the specifications compared to the control flow based counterpart. In control flow based CDFG there is almost one-to-one mapping between the nodes of the CDFG and the lines of HDL code. If there are nested loops and conditions in the input specifications then it should be flattened before extracting the data flow based CDFG. Further, in such cases the CDFG may have a large number of nodes.

In CDFG the data flow CDFG this is tedious to draw from specification because this no one to one mapping. So, in this case we could have drawn now here in nesting loop. So, you could have done this one by a simple loop is that think it is a nested loops, if there are procedurals, if there is sometime inside a while loop, which is a for loop like inside there is some kind of procedural call and some external variable call etc. are there. So, then it will be very difficult to go for this data based CDFG from the Verilog. Then you have to flatten the design and make it non nested based stuff to do that.

Ok. So, this there is a lot of manual intervention. So, in control flow based CDFG only one to one mapping between the nodes and CDFG, but in case of nested loop, conditions etcetera then everything is to be flattened. Before you can go for a data flow based CDFG, but in case of control based CDFG it is very simple. It is just a one to one mapping. So, you can just have a if it a nested loop so you have lateral composition block inside, we will have squarely computation block and so forth, because it is one inside the other.

This is possible in control flow based CDFG , but in the data flow based CDFG as you have to eliminate redundancies and all, you have to first eliminate the what you call this nested stuff. It have to be flattened and then only you can use some more manual intervention tools, you have to convert it from the Verilog code to a data flow based CDFG. So, even if data flow based CDFG is more near to hardware and so forth. But

still the redundancies are not there, but still it is a difficult problem because we have to flatten and then do some manual intervention and or you have to do some computational I should say not manual intervention but the computation you should do while in the case of control flow it is just a one to one mapping.

Ok. With this we come to end of this lecture. Now still now we have seen, what we have seen? We have seen that we started with hardware, HDL what do we call high level synthesis. Now in hard level synthesis, what you have seen today? You have seen how can you give the input specification and how can we represent your input specification in terms of control and data flow graph. In the next lecture so what we will see, how can you transform using this data flow graphs? How can you go from transformation tone form to other like we have to transform the input specification to high level designs, high level architecture designs. So, how easy it is. To transform them from the simple specification in CDFG form to high level I mean what you call these architectural blocks using this CDFG, that we will see in the next lecture.

Thank you.

.