

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

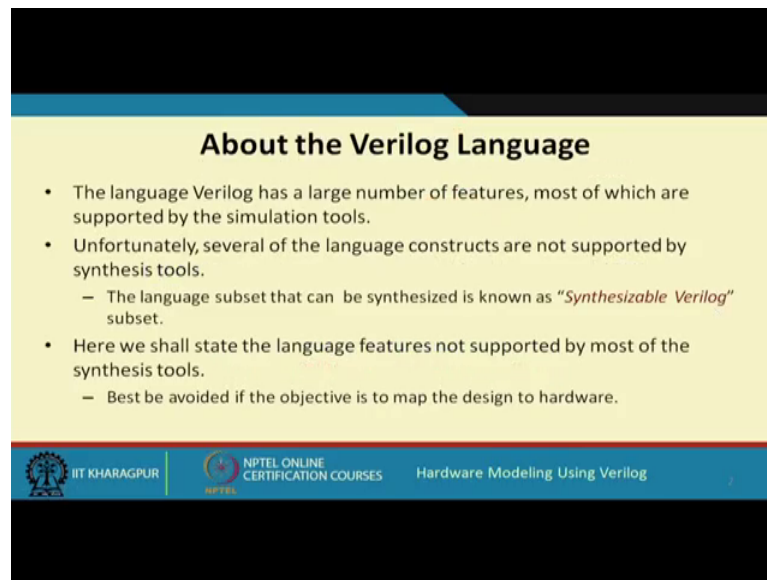
Lecture - 28
Synthesizable Verilog

So, in the lectures so far we have seen various ways to model both combinational and sequential circuits. So, if you recall we talked about and also learnt how to model. So, called behavioral specifications of some digital system block and also we learnt how to model some designs in a structured way structured design. So, broadly speaking designs can be categorized into behavioral and structural and one thing.

So far whatever results that we discussed we showed they were based on simulation only, but in reality whenever you are trying to design a hardware block I mean either on a p j or on a 6 then you will have to use some kind of synthesis tool. Simulation can only be a first step to carry out initial verification of a design, but when you are doing synthesis you may see that a lot means other problems are cropping up there are some some errors which I showing which were not reflected during the simulation phase.

So, in this lecture we shall be talking about some of the features of the verilog languages that are meant to be used for synthesis, the other features which we have discussed in the class, but they are often not accepted by a synthesis tool if you use those constructs the synthesis tool will not be able to generate the final hardware circuit or the net list ok.

(Refer Slide Time: 02:18)



About the Verilog Language

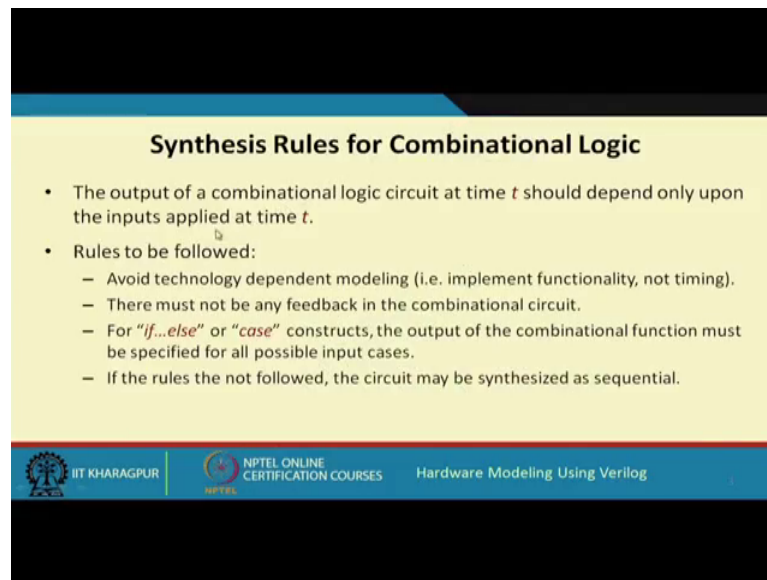
- The language Verilog has a large number of features, most of which are supported by the simulation tools.
- Unfortunately, several of the language constructs are not supported by synthesis tools.
 - The language subset that can be synthesized is known as “*Synthesizable Verilog*” subset.
- Here we shall state the language features not supported by most of the synthesis tools.
 - Best be avoided if the objective is to map the design to hardware.

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

So, the title of today's talk is synthesizable verilog. So, whatever I had just talked about just now. So, the verilog language provides you with the number of facilities and features and this features are mostly supported by the simulation tool for example, the simulation tool that that we have been using a part of this course the I verilog and of course, the waveform viewer g t k wave they are all known to support most of the verilog constructs that we have been discussing throughout the class right. But as I had said there are some language constructs which are not accepted by the synthesis tools this subset of the language verilog which are actually accepted by the synthesis tools are referred to as synthesizable verilog subset.

Now, in this lecture we shall be looking at some of the language features which are not supported and some of the recommended styles of modeling for synthesis of circuits both combinational and sequential.

(Refer Slide Time: 03:36)



Synthesis Rules for Combinational Logic

- The output of a combinational logic circuit at time t should depend only upon the inputs applied at time t .
- Rules to be followed:
 - Avoid technology dependent modeling (i.e. implement functionality, not timing).
 - There must not be any feedback in the combinational circuit.
 - For “if...else” or “case” constructs, the output of the combinational function must be specified for all possible input cases.
 - If the rules are not followed, the circuit may be synthesized as sequential.

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

So, talking about the synthesis rules for combinational logic you recall what do mean by a combinational logic, a combinational logic is a hardware circuit which does not have any kind of enough fan outs or storage devices built into it.

So, the output of the circuit will only depend on the currently applied inputs well of course, the circuit the gates will have some delays they output will be available just after the circuit delays, there is no notion of clock and other kind of synchronization that coming in case of combinational circuits

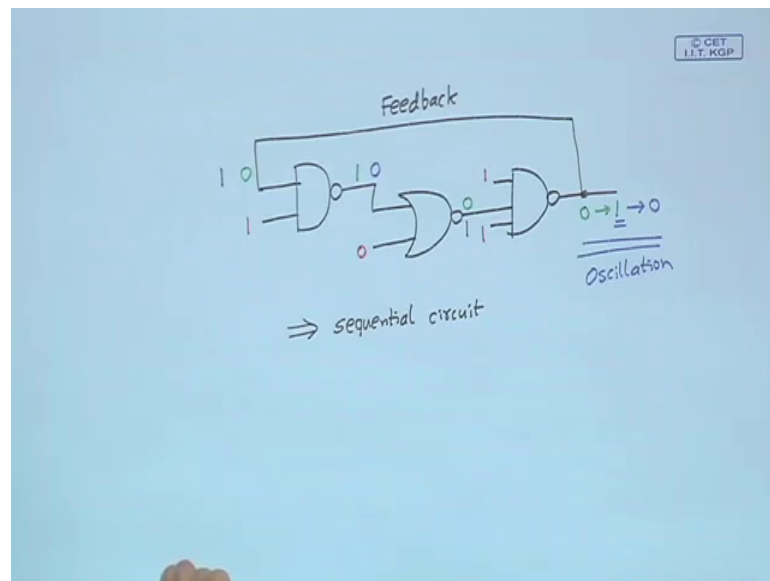
So, you can say that the output of a combinational circuit at some point in time let us say t should depend only upon the inputs that I have applied at that time of course, there will be a delay of the circuit that we are not mentioning here well. So, we were trying to model combinational logic there are some rules that need to be followed, first thing is that since you are talking about combinational logic do not use I means any kind of technology dependent modeling means specifically some details about timings see the timing detail using that hash command.

So, whatever you give during simulation that is just meant for carrying out the simulation and interpreting the waveforms that you are viewing after the process of simulation is over, but when you are synthesizing you are finale delays will be dependent on the actual hardware. So, where your mapping it can be a esic or a p j. So, you do not have any control about the delay you cannot set the delays one or 2 or 3 nanosecond or picosecond

you cannot say that that will be entirely depend on the hardware where you are finally, going to map your design tool ok.

So, this kind of technology dependent modeling are not allowed when you are trying to synthesize combinational circuits. Secondly, of course, in a combinational circuit there are no feedback. So, your net list or your circuit must not have any feedback, feedback means some connection from the output of a circuit to the input of a circuit.

(Refer Slide Time: 06:20)



Let us take an example suppose I have a circuit like this, there are other inputs suppose what I say is that this output line is connected to one of the inputs this is what you mean by feedback. Well feedback is not allowed in a combinational circuit for obvious reasons because when your feedback your circuit may turn in to a sequential circuit right.

So, in this example let us say I mean we have applied a one here we have applied a 0 here and a 1 and 1 here. So, now, what will happen what will be the behavior of the circuit? Let suppose that initially the output of this of this nand gate is 0. So, 0 is being feedback this is 0 0 and 1 nand will be 1, 1 and 0 nor will be 0, 0 1 and 1 nand gate. So, the output will be changing to 1 right.

Now, when the output is changing to 1 this 1 will again be feedback, now the new input will be 1, 1 and 1 will be now 0, 0 and 0 will be now 1, 1 1 1 will be again 0. So, there

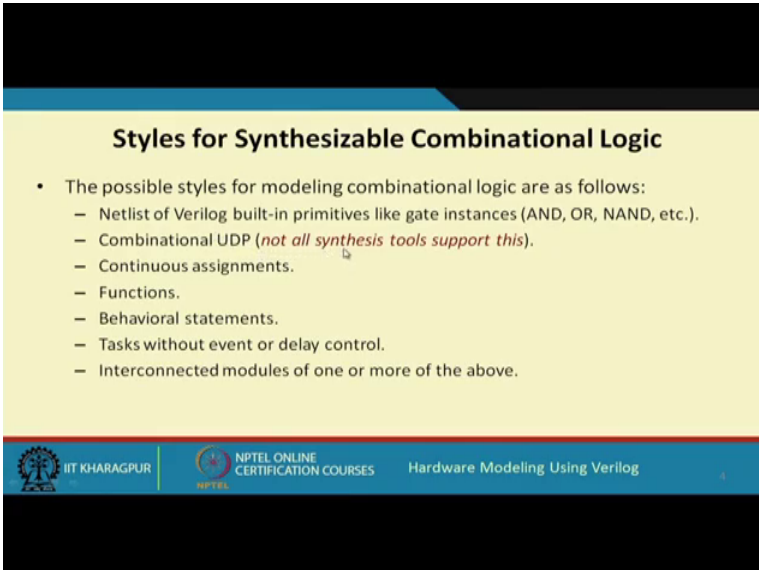
will be something called oscillation in the output it will continuously go from 0 1 again to 0 again to one again to 0. So, the output will never stabilize

So, if your object is to design a combinational circuit you must avoid feedback connections like this, there must not be any kind of feedback in your net list or circuit right and the third point I mentioned repeatedly earlier that whenever you are having a some kind of multi way branching. So, either using if else or using a case construct while we are checking for some condition and then you are deciding on something some assignments.

So, here the output of the function whatever you are assigning to must be specified for all possible input cases like for instance if you are checking for, for a condition which is a 2 bit variable then you must specify what the output will be for all 4 combinations of those 2 bit variables for 0 0, 0 1, 1 0 and also 1 1.

So, we discussed earlier that if you forget to specify one of the input combinations then the synthesis tool will be generating a latch for the output. So, even if you are trying to generate a sequential circuit. So, whatever will get generated will be sorry means actually you are trying to generate a combinational circuit, but whatever will be generated that will be a sequential circuit with a storage element. So, you must avoid this kind of incomplete specification in multi wave branching.

(Refer Slide Time: 09:56)



Styles for Synthesizable Combinational Logic

- The possible styles for modeling combinational logic are as follows:
 - Netlist of Verilog built-in primitives like gate instances (AND, OR, NAND, etc.).
 - Combinational UDP (*not all synthesis tools support this*).
 - Continuous assignments.
 - Functions.
 - Behavioral statements.
 - Tasks without event or delay control.
 - Interconnected modules of one or more of the above.

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

So, I means as I said if you do not follow these rules the circuit may become sequential. Here are some styles that you need to follow for the synthesis of combinational logic, some of this we have already seen some of this we have not talked about earlier we shall just look at them through some examples like the possible styles maybe. Firstly, we have seen several examples. So, you can instantiate the basic primitives of gates and or nand x or and so on and you can create a net list like that suddenly you can use user defined primitive for combinational circuit you can define a truth table, right. Most of the synthesis tool support combinational u d ps, but you need to check there. There are some synthesis tools which will not accept any kind of u d ps at all even if it is a combinational u d p with a truth table ok

So, that is mentioned not all synthesis tools will support this feature then continuous assignment using assign statements this of course, you can use for modeling combinational logic and functions at something which I have not talked about earlier. These functions are something where I means you can use this continuous assignment statements in conjunction with the functions we will see, we will see that there some advantages or the code becomes much easier and more structured.

Then of course, we can use the always block for modeling combinational circuit we have seen many examples here behavioral statements and just like functions there is another thing called tasks. So, we can use tasks as well, but we cannot specify any kind of delays and you can use a net list of the modules created using one or more of these you can combine means any number of them you can create a some kind of interconnection of them by using instantiation.

(Refer Slide Time: 12:15)

(a) As a Gate Netlist

```
module example (x1, x2, x3, x4, y);
  input x1, x2, x3, x4;
  output y;
  wire w1, w2, w3;
  or (w1, x1, x2);
  or (w2, x3, x4);
  xor (w3, x3, x4);
  nand (y, w1, w2, w3);
endmodule
```

Shall be synthesized in terms of gates from some target technology library.
The gate netlist is often optimized during synthesis.

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

Let us look at these examples some of this one by one. So, as a gate net list so we have seen many examples earlier this is a very simple example there are 5 parameters these 4 are the inputs and y is the output and this y 1, y this w 1, w 2 w 3 are some intermediate wires. So, here you are specifying that there are 4 gates or x 1, x 2 is input w 1 is the output this is second or xor and nand. So, in this way you can create any kind of net list.

Now, the point to note is that here we are directly specifying the gate types, but this synthesis tool when it is synthesizing it will be generating the final hardware right. So, the final hardware will be generated based on some target library, let say for example, the target library does not contain any exclusive or gate. So, the x or gate which you have specified here they will have to be generated using different kinds of gates maybe a combinational nand gates or and or not gates and so on. So, during the synthesis process whatever gates you have specified there can be some changes as well because you will have to use gates only from the target technology library that is called; that means, the target gates which are supported by the hardware. So, during the process the synthesis tool also carries out some kind of optimization ok.

So, these are done here. So, when you are carrying out this mapping some of the gate level minimization algorithms can be used.

(Refer Slide Time: 14:16)

(b) Using Continuous Assignment

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

Shall be mapped to gates or cells from some target technology library.
The Boolean equations are optimized during synthesis.

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

Next continuous assignment is something which also we have seen through many examples, here we can use an assign statement to assign some Boolean expression to a variable which is of type where. Now, this is a very simple example a module which computes the carry output of a full adder so it is a b or b c or c a right.

So, here also we have not specified any gates here we are specifying just the function. So, the so the synthesis tool again will take the functional specification as the input and will be trying to generate some gates or cells from the target technology library after some kind of minimization or a optimization, this is what will be done if we use the assignment statement.

(Refer Slide Time: 15:16)

(c) Using Procedural Blocking Assignment

```
module mux2to1 (f, in0, in1, sel);
  input in0, in1, sel;
  output reg f;

  always @(in0 or in1 or sel)
    if (sel) f = in1;
    else f = in0;
endmodule
```

Inputs to the behavior (*here in0, in1, sel*) must be included in the event control expression; otherwise, a latch will be inferred at the output.

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

Similarly, you can model combinational circuits using procedural block assignment where the event is not an h regard event, this is an example of a simple example of a 2 line to 1 line multiplexer where in 0 and in 1 are the inputs sel is the select line and f is the output. So, these 3 are the inputs and f is the output which is also reg because you are assigning f within a procedure block.

So, here we are saying always whenever any of the inputs are changing if the select line is 1 then in one is selected to f otherwise in 0 is selected to f . So, the point to notice that when you are specifying a combinational logic like this your activity list must contain all the inputs in this case the inputs are in 0, in 2 and sel. So, all the inputs of the behavior which are used in this procedural block must be included in the event control because if you do not do it this synthesis tool will assume that well those are not the inputs, but still you are using it. So, if the inputs are not specified you do not know you may have to use a latch to store the output f. So, latch will often be inferred by the selection tool if you give incomplete input list specification. So, make sure that you give a complete input list specification in this always block ok.

(Refer Slide Time: 16:53)

(d) Using Functions in Verilog


```
module fulladder (s, cout, a, b, cin);
  input a, b, cin;
  output s, cout;
  assign s = sum(a, b, cin);
  assign cout = carry(a, b, cin);
endmodule
```

```
function sum;
  input x, y, z;
  begin
    sum = x ^ y ^ z;
  end
endfunction
```

```
function carry;
  input x, y, z;
  begin
    carry = (x&y) | (y&z) | (z&x);
  end
endfunction
```

A function in Verilog returns a single value.
Can be used to make a code more readable.
Typically used with "assign".
Input arguments appear in same order.

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



Now, let us see how functions work we have not talked about functions earlier, see function is very similar to a function in high level language like c, c which means in a language like c you can have functions. Now, now in a function you can pass a number of variables through the parameters, but you can pass a single result through the name of the function. So, here also whenever you are using a function it is intended to have a single output a single bit output. So, it cannot have more than 1 outputs this is a restriction of function in verilog.

So, let us say here we are trying to write a module for a full adder just using assign statement, but instead of directly writing down the expression we are calling 2 function sum and carry, these 2 functions we have written separately. Function the syntax is like this function followed by the name sum is the name, then I will have to specify a list of the inputs and they have to be provided in the same order you are mentioning them here a b and carry in. So, a will be map to x, b will be map y and c in will be map to z and then here you are giving an assignment statement for the left hand side is a name of the function and right hand side is any expression based on the this input variables this is the sum function similarly this is the carry function.

So, you can directly call this functions in the main full adder module what is the advantage is that your module becomes easily understandable, it is it is much more well documented. So, instead of writing the expressions here this module will become more

clumsy rather than if you write like this it will be much clearer. So, as I had said so function in verilog will return a single value here sum, here carry and the value will be return against this expression whenever you are writing an expression on the right hand side. So, whatever is the calculated value of sum that will be assigned to s, similarly for the carry function the value will be assign to c out. So, this makes the code more readable as I said and these are typically used with assigned as this example illustrates and also I have shown that the input arguments must appear in the same order in the function and the place where you are calling them.

(Refer Slide Time: 19:46)

```
module fulladder (s, cout, a, b, cin);
input a, b, cin;
output reg s, cout;

always @(a or b or cin)
    FA (s, cout, a, b, cin);

task FA;
output sum, carry;
input A, B, C;
begin
    #2 sum = A ^ B ^ C;
    carry = (A&B) | (B&C) | (C&A);
end
endtask
endmodule
```

(e) Using Tasks

The arguments must be specified in the same order as they appear in the task declaration.

More than one output value can be returned.

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

Now, let us come to tasks well a task is more general before going on the example let us see what a task, is the first thing about task is that using task, task is like a function, but you can pass more than one output value to the calling program or the module like you see, an example and you see for function you can use it only inside assign as I had said, but a task can be used anywhere task can be used anywhere inside a verilog function you write a task and you can call it from someplace.

So, wherever you have defined the task that will get substituted in the place where your calling with proper argument passing, let us take this example. So, this is again a full adder example. So, what you are saying we are using the behavioral modeling seen always at a or b or c in whenever the input change well instead of writing the sum and carry expressions we are calling a task.

Now, if a is a task we have defined f a within bracket the list of parameters, s carry out a b c in. So, in this list of parameters 3 of them are inputs and 2 of them are outputs so s and c out will be returned. Now the way task is defined is like this task name and end task and inside this we will have to specify the outputs and the inputs in the same order as they appear here, let s and a out are the 2 outputs you specify them maybe the names will be different here we call it them sum and carry then a b c in they are the inputs. So, here you call them a b c, now in task you see you can also specify a delay which you cannot in a function. So, some expression, carry expression. So, this is how we can use a task in a function..

Now, in our earlier examples which we have seen we have not given any example that uses function or task, but if you want for a larger designs you can use functions and tasks as well fine. Now let us look at the difference between function and a task so, what are the main difference and where you can use a function and where you can use a task right.

(Refer Slide Time: 22:33)

Difference between Function and Task	
Function	Task
A function can call another function but not another task.	A task can call other tasks and functions.
A function executes in 0 simulation time.	A task may execute in nonzero simulation time.
A function cannot contain any delay, event, or timing control statement.	A task can contain delay, event, or timing control statements.
A function always return a single value.	A task can pass multiple values through "output" and "inout" type arguments.
A function must have at least one input argument.	A task can have zero or more arguments of type "input", "output", or "inout".

So, let us see this table s o, a function first thing is that. So, a function can call another function, but a function cannot call a task. So, you can have multiple functions and you can have nested calls a function x can call a function y y can call a function z just like in a high level language we do this is how function calls are made, but tasks are more general. So, a task can either call a function or it can call some other task.

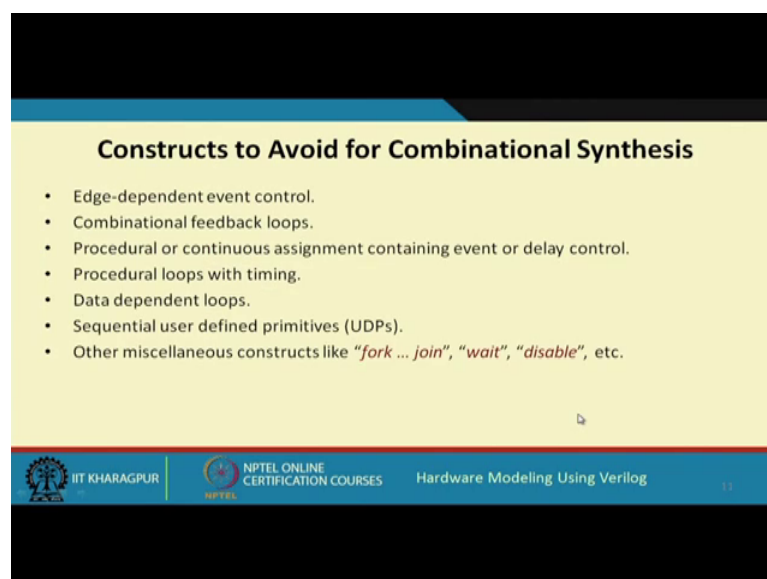
Now, in a function the concept of delay is not there. So, it is assumed that a function will execute in time 0.

Student: (Refer Time: 23:21).

But, in the task in the example that I have shown it also shows there that you can also include some delays. So, it can execute in non 0 simulation time and a function just usually contains only combinational assignments nothing else, but in a task it can be more general you can contain delay like the example showed, you can contain some event triggering or some timing control statements as well the function returns a single value this is another thing and a task it may return more than one value through output arguments, like in the example that I have showed earlier there were 2 output arguments sum and carry.

Now, another kind of port declaration is there which I have deliberately not talked about I shall take some examples later these are some kind of bidirectional inputs in out, well in out means you can use that variable as an input and also as an output, but when you map it to the hardware sometimes some problems are created for in out type variables that is why in the designs that I have shown I have avoided in out fine and another thing is that a function must have at least 1 input argument, but there is no such restriction in task it can have 0 or more arguments and the arguments can be of type either input or output or in out.

(Refer Slide Time: 25:01)



Constructs to Avoid for Combinational Synthesis

- Edge-dependent event control.
- Combinational feedback loops.
- Procedural or continuous assignment containing event or delay control.
- Procedural loops with timing.
- Data dependent loops.
- Sequential user defined primitives (UDPs).
- Other miscellaneous constructs like *"fork ... join"*, *"wait"*, *"disable"*, etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 11

Now, these are some of the constructs which you must avoid when your objective is to carry out combinational circuit synthesis like we should not give this edge dependent even control like you must not give means always at posedge clock or neg edge clock no kind of posedge or neg edge kind of event control should be there in a combinational circuit description.

Then second thing I already mentioned there must not be any feedback loops. So, when you specify net list using instantiations make sure that you do not include any feedback loops, third thing is that you can have procedural assignment or continuous assignment, but there must not be any delay specification or event control, because in synthesis those are not considered.

Similarly, you can have a loop with timing there can be some delay specified. So, the delays are not allowed then data dependent loops, data dependent loop means some kind of a loop where the number of times you are going to loop it depends on a variable that kind of thing is not allowed. The number of times you are looping must be a constant. So, if it is a constant suppose 3 I specify I want to execute a for loop 3 times then the synthesis tool will make 3 copies of whatever is the body of the loop it will make 3 copies of the circuit and it will create a combinational circuit like that there will be 3 levels which corresponds to the 3 iterations of the loop right.

Sequential user defined primitives like state table or not permitted by synthesis tools and there are some other miscellaneous constructs which I have not discussed deliberately like fork join. These are used to specify concurrency wait well wait is used sometimes to make a statement wait for certain time before it gets activated and of course, disable you can disable some signals are commands.

(Refer Slide Time: 27:27)

Summary: Synthesizable Verilog Constructs

- *"module ... endmodule"*
- Instantiation of a synthesizable module
- *"always"* construct
- *"assign"* statements
- Built-in gate primitives
- User defined primitives – combinational only
- *"parameter"* statement
- *"functions"* and *"tasks"*
- *"for"* loop
- Almost all operators
- Blocking and non-blocking assignments
- *"if ... else"*, *"case"*, *"casex"* and *"casez"*
- Bits and part select of vectors

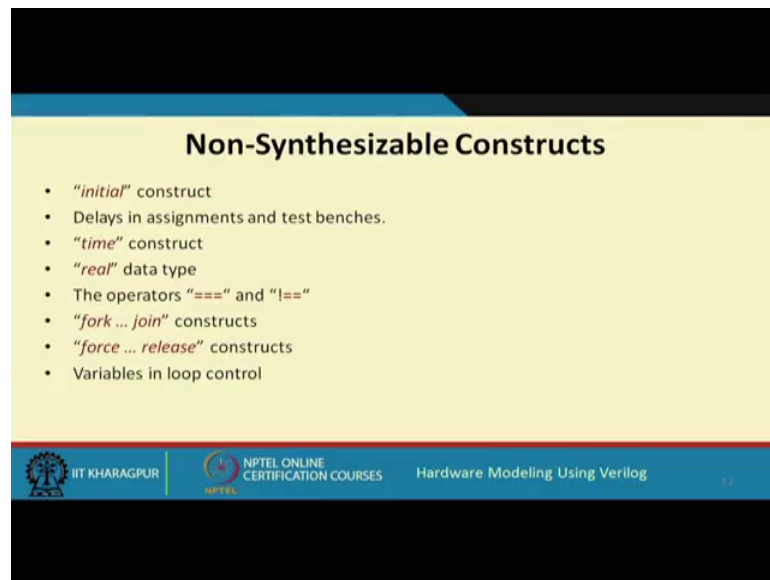
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 12

So, to summarize this synthesizable Verilog constructs the construct that you must use when your objective is to synthesize the design into some hardware, there as follows module end module of course, well you are allowed to carry out instantiation you can use always constructs assign statements built in gate primitives like and or not x or, nand, nor this you can use, but for user defined primitives you can possibly use only combination specification, but still you will have to check with the synthesis tool your using that whether the tool supports combination u d p or not. Parameters you can use functions and tasks as I have mentioned as something that can be used for loop is usually supported by all synthesis tools the other kind of loops may not be supported.

So, again we will have to specifically check whether the other kind of loops like while or repeat they are supported or not. Regarding operators almost all the operators are supported by adding a couple of them a very few blocking and non-blocking assignments are supported and multi way branching using if else case, case x, case z. So, you can use bits and with respect to vectors the part selection you can select some segments from a vector those are supported ok.

So, if you stick to these rules that mean you will not be using anything outside this then it is often guaranteed that whatever module that you write that can be synthesized into a hardware by the synthesis tool.

(Refer Slide Time: 29:22)



Non-Synthesizable Constructs

- *"initial"* construct
- Delays in assignments and test benches.
- *"time"* construct
- *"real"* data type
- The operators *"==="* and *"!=="*
- *"fork ... join"* constructs
- *"force ... release"* constructs
- Variables in loop control

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Hardware Modeling Using Verilog | 17

And there are a few things which are not synthesizable definitely like initial construct it is meant only to be used in the test benches delay specification is in the hatch command.

So, you cannot use delays in synthesizable code, time no concept of time in synthesizable code, real data type is often not supported, only bits and integers they are supported and the operators this triple equal and not equal to equal to these 2 are not supported fork join force release and as I had said in loop control you cannot loop variable number of time. So, variables in loop control are typically not supported.

So, with this we come to the end of this lecture. So, in this lecture we try to give you just an idea that what are the constructs in verilog that you have learned. So, far which are useful for synthesis and what are the constructs which are best avoided because most of the synthesis tools do not support that.

Now, in the next lecture we shall be continuing with our discussion, we shall be talking about some recommended practices some dos and do nots you see verilog maybe supporting a few things. So, the synthesis tool may be supporting many things, but if you go to an industry who is specialized or whose specializes in design they will give you some guidelines the designer will have to stick to those guidelines whenever they are designing any circuit using some language like verilog, because if they do that then the codes will be well documented they can be reused and they can be maintainable.

Thank you.