Hardware Modeling using Verilog

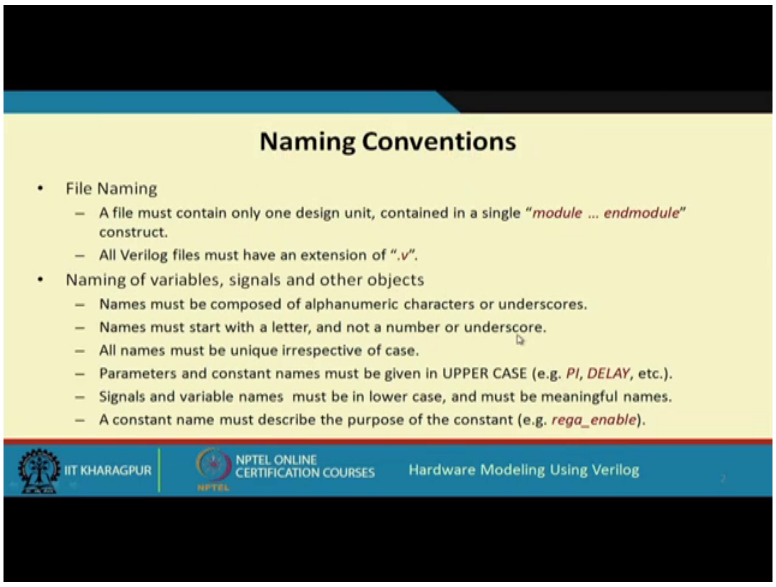
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 29
Some Recommended Practices

So, in this lecture we shall be discussing some of the recommended practices that are typically followed in an industry which works in the area of digital design. So, it is good to learn this, because some of you who may be going to the industry and work in the design area in the future. So, you will be encountering these kinds of guidelines there. So, every industry they have a set of guideline, where they specify that what are the things that you should do, and what are the things you should avoid when you are creating a design, using some hardware description language, ok.

So, let us see some of these recommended practices.

(Refer Slide Time: 01:04)



Naming Conventions

- File Naming
  - A file must contain only one design unit, contained in a single "*module … endmodule*" construct.
  - All Verilog files must have an extension of "*.v*".
- Naming of variables, signals and other objects
  - Names must be composed of alphanumeric characters or underscores.
  - Names must start with a letter, and not a number or underscore.
  - All names must be unique irrespective of case.
  - Parameters and constant names must be given in UPPER CASE (e.g. *PI*, *DELAY*, etc.).
  - Signals and variable names must be in lower case, and must be meaningful names.
  - A constant name must describe the purpose of the constant (e.g. *rega_enable*).

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

First thing concerns the naming conventions. These concerns file naming. Now see Verilog compiler simulator or synthesizer whatever you call, they support filenames, many of them they support file names, which can have arbitrary extensions. Not necessarily dot v or dot Verilog or anything, you can have any extension dot anything.

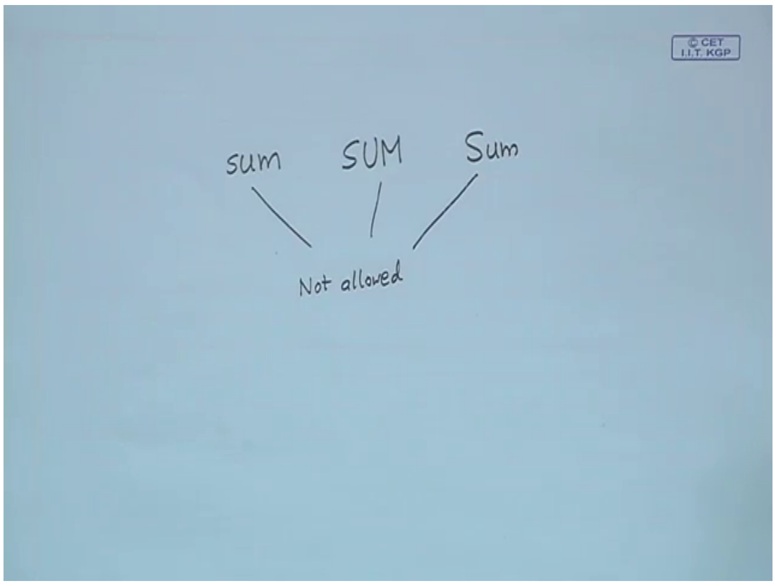
But let us say here we are specifying a set of typical conventions. So, here we are saying that all the files must have an extension of dot v, and we must not include more than 1 module in a file, this is 1 rule that you should follow.

A file must contain only 1 design unit. You see this does not mean that you cannot write more than 1 modules in a file. Yes you can do, you can write 10 modules in a file you can synthesize, you can simulate, but these are some of the design practices, which is practiced in typical industry houses ok.

Regarding the namings of the variables signals or any other object, the conventions are as follows, names should be composed of alphanumeric characters or underscore. Alphanumeric means alphabets numerics and underscores. Some verilog synthesis tools are simulators, they support other, few other characters also, but this guideline says that you should avoid those, stick to alphanumeric and underscores only right, and again even if the compiler supports. So, every name must start with a letter, you should not begin a name with an underscore.

Third point says the names must be unique irrespective of case. So, what does this mean. This means Verilog is case sensitive, where uppercase and lowercase letters are considered to be different. So, you must not do something like this.

(Refer Slide Time: 03:40)



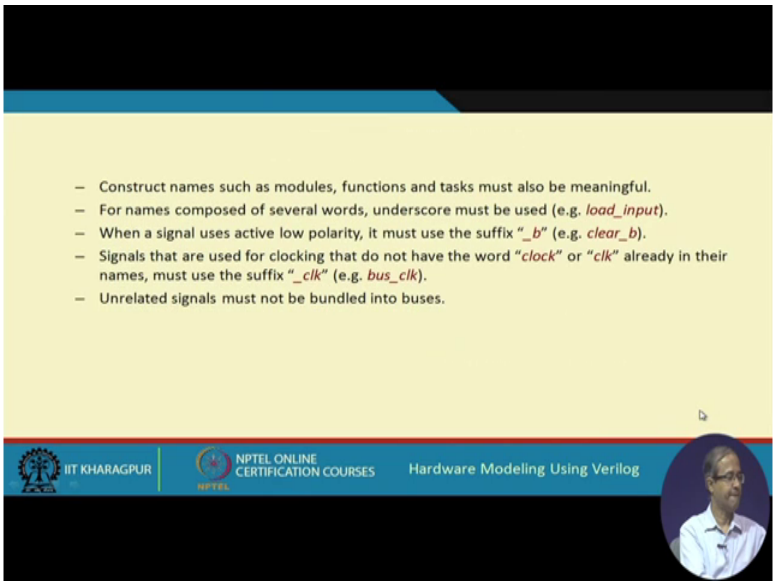
Let us say you define 1 variable as lowercase S U M sum, define another variable as uppercase S U M sum, and you define another variable which is a combination of this. So, this guideline says that this is not allowed. So, again this is not allowed, it does not mean not allowed by the compiler, it is not allowed by the company, something like this right.

So, some other conventions it says parameter and constant names. So, whenever you are defining some constants. So, either by using some defined statement or using parameters we have already seen, the names must be given in uppercase. For example, pi p I d e l a y. So, that by just seeing a variable name you will understand that whether it refers to a normal variable reg type or wire type, or it is a constant or a parameter, just by seeing the variable you can understand.

Similarly, signals and variable names, they will be in lowercase. Not only that names should be meaningful this is quite obvious. Instead of giving names like a b c d e f, you should give names like carry sum out like that. So, that the names are somewhat meaningful in the context of the design that you are trying to create.

Then it says that a constant name must describe the purpose of the constant, whenever you are defining some constant or a variable name, like see reg a underscore enable something like that this can be very well, because a constant I have said that it will be uppercase. So, it can be anything uppercase or lowercase, but it should be meaningful.

(Refer Slide Time: 05:43)



- Construct names such as modules, functions and tasks must also be meaningful.
- For names composed of several words, underscore must be used (e.g. *load_input*).
- When a signal uses active low polarity, it must use the suffix "*_b*" (e.g. *clear_b*).
- Signals that are used for clocking that do not have the word "*clock*" or "*clk*" already in their names, must use the suffix "*_clk*" (e.g. *bus_clk*).
- Unrelated signals must not be bundled into buses.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

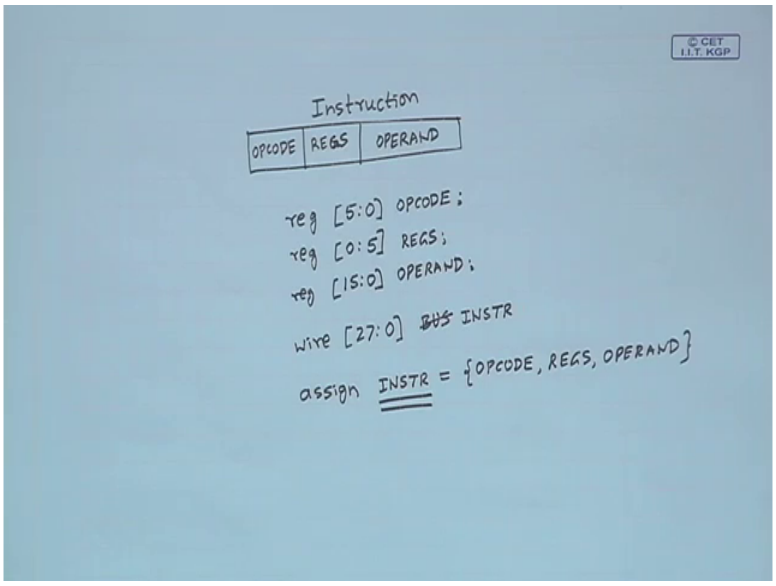
So, this repeats not only the variables, names of the modules, the functions and the tasks they also must be meaningful. So, that just by seeing that module or function or task, you can have a rough idea what it is trying to do right. So, if a name is a little long, compose of several words we can use underscores, like this load underscore input, this convention can be followed.

Now, see signals can be either active high or active low. let us take an example, suppose I have a register, there is a clear input. So, active high means, whenever clear is 1 the register will be cleared. Active low means whenever that clear is zero, the register be cleared. So, in a design I can use many such signals; some of them can be active high some of them can be active low. Just as a matter of convention in order to identify which of the signals are active low, this convention says that you use a suffix underscore b for the active low polarity signals. Like clear underscore b. This will tell you that clear is a active low signal, b is a short form for below ok.

Signals that represent clocks, typically we give names like c l k or clock, but there can be some other signals also. They represent clocks or they are derived from the clock, using something called clock gating. Like you can take a clock signal you can take some other signal line you can take an AND gate, and output will be a. So, called gated clock that gated clock can also be used to control or activate some storage cells or registers ok.

So, such signals you should give an explicit name by suffixing it underscore c l k, like if the name of the signal is bus you write bus underscore c l k, indicating that this is also a clock signal. And the last point is quiet obvious unrelated signals must not be bundled into buses. Like let us say, let us take an example.

(Refer Slide Time: 08:15)



Instruction

OPCODE	REGS	OPERAND

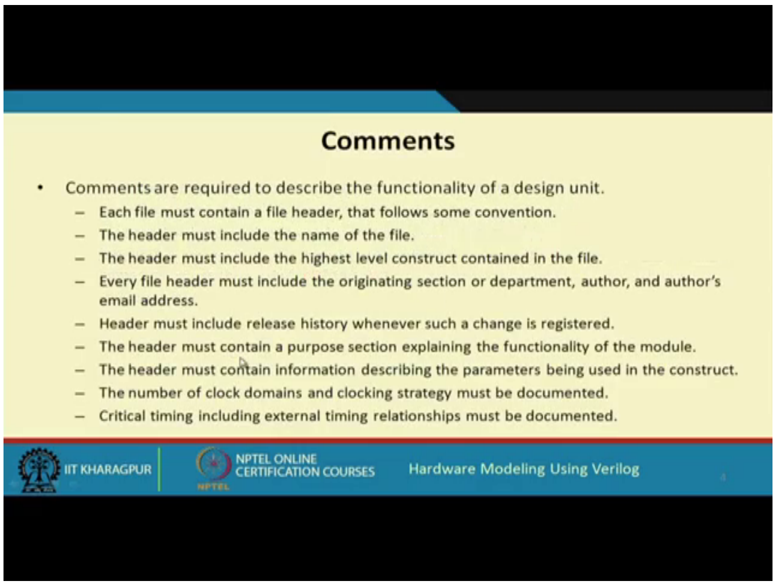
```
reg [5:0] OPCODE ;
reg [0:5] REGS ;
reg [15:0] OPERAND ;
wire [27:0] ~~BUS~~ INSTR
assign INSTR = {OPCODE, REGS, OPERAND}
```

Suppose I have a word that indicates an instruction in a processor. So, in this instruction, I can have an OPCODE. So, I can have some fields that indicate registers, registers and some fields that indicate OPERAND. So, I can, what I can do, I can declare them separately like if OPCODE is a 6 bit quantity, I can declare them like this. If REGS is a, let us say, this is our 6 bit. See here I am writing different way just to show you that I can do this also, register and this OPERAND, let us say this is a sixteen bit quantity let us say.

Now, what I am saying is that, you can specify a bus, let us say this is how much 5 6 and 6 12, and 1628. So, let us have a 28 bit bus, let us call it bus. So, I can always give a assign statement like this, bus or you can give any other meaningful. Instead of bus let us say let us give a name instruction, this will be more meaningful. So, assign instruction equal to composition of OPCODE, REGS, OPERAND. This means 3 different things, and grouping them together and I am referring it to a single entity this instruction.

Here in this point, means exactly the same thing is mentioned. It says that whenever you are grouping like this, the signals you are grouping must be related. Like here OPCODE registers and OPERAND, they are all part of the same instruction; that is why you are grouping them, but unrelated signals you should never group fine.

(Refer Slide Time: 10:44)



Comments

- Comments are required to describe the functionality of a design unit.
  - Each file must contain a file header, that follows some convention.
  - The header must include the name of the file.
  - The header must include the highest level construct contained in the file.
  - Every file header must include the originating section or department, author, and author's email address.
  - Header must include release history whenever such a change is registered.
  - The header must contain a purpose section explaining the functionality of the module.
  - The header must contain information describing the parameters being used in the construct.
  - The number of clock domains and clocking strategy must be documented.
  - Critical timing including external timing relationships must be documented.

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

Next this is very important in a design comments. So, if you write a Verilog code which is functionally correct, everything is fine, but there are no comments, then after sometime you will see that, you yourself will not be able to understand that code you forget about others. So, it is mandatory to have every piece of code very well documented, and that is done using comments.

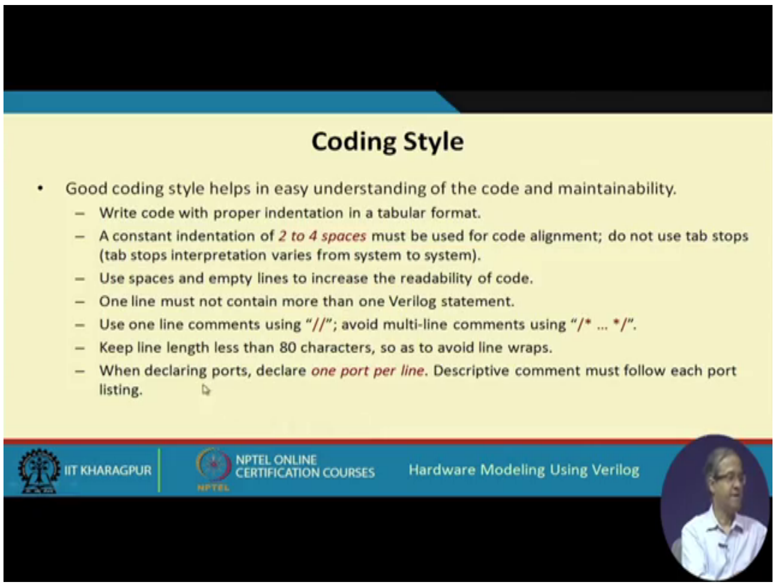
Now, some of the conventions regarding comments are as follows, is at every file you create must contain a header, that contains lot of information, and every company has its own contingence. I will give an example later. So, the header among other things must also include the name of the file; such that accidently the name should not get changed right.

So, it should also contain the highest level construct contain in a file, because you see, just an Verilog you can create a design in a hierarchical way. Like we took an example of an adder a ripple carry adder, built using full adder, a full adder built using sum and carry and so on. So, that header should also specify what is the level of hierarchy that module or that file is specifying or referring to fine.

And of course, it must contain some descriptive note that what the module is actually doing. So, I am explain the functionality of the module, and the parameters used in the module, what are the functions of the parameters, and if there are multiple clock domains you are using multiple clocks, may be a slow clock or fast clock, then the header must also specify. I mean how many clock domains are used and what kind of clocking strategies used, some may be leading which triggered, some may be falling is triggered and some issues regarding critical timing may also be documented.

Because say for example, when you are designing a module, may be you know that there is 1 signal, which falls in the critical path. If you make that signal generation slower your whole system will slow down. So, let that remain documented in the header. So, that in the future when someone updates or modifies that module, you will know that well this is a very important signal, I must not make that signal slower; that is a critical signal fine.

(Refer Slide Time: 13:37)



Coding Style

- Good coding style helps in easy understanding of the code and maintainability.
  - Write code with proper indentation in a tabular format.
  - A constant indentation of *2 to 4 spaces* must be used for code alignment; do not use tab stops (tab stops interpretation varies from system to system).
  - Use spaces and empty lines to increase the readability of code.
  - One line must not contain more than one Verilog statement.
  - Use one line comments using "//"; avoid multi-line comments using "/* ... */".
  - Keep line length less than 80 characters, so as to avoid line wraps.
  - When declaring ports, declare *one port per line*. Descriptive comment must follow each port listing.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

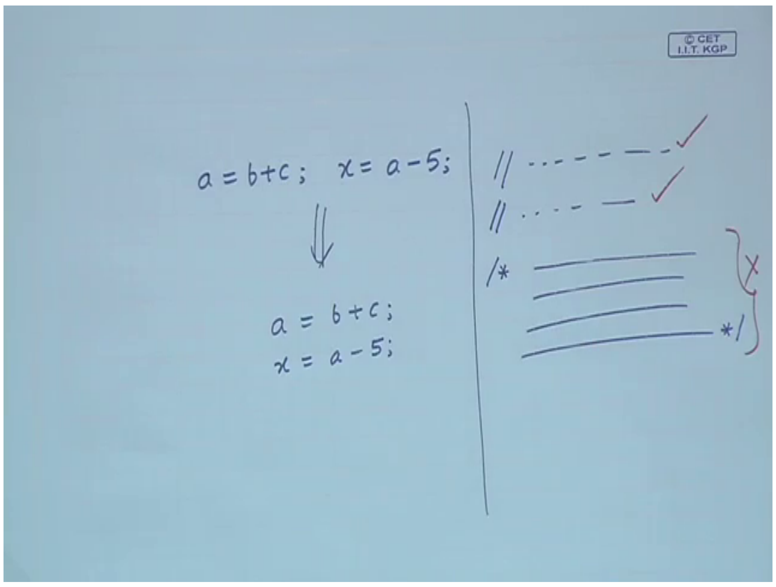
Some of the coding style should also be followed, this will of course, make the code more easily understandable and it will be easier to maintain by others mostly. Firstly, is that you must give indentation in the code indentation, means proper spacing in a proper tabular format. So, again I will give an example, we see earlier all the examples that I have shown their indentation was used. So, I never showed you all the instructions starting from the same column, there will be some spaces, again some spaces, again some spaces.

So, just the position of the statement will tell you that under which block that statement is belonging to. So, indentation gives you a very nice visual structure of your whole code right. Not only indentations. Well this indentations are typically this guideline says you must give 2 to 4 spaces for indentation well, and there is also a recommendation, it says do not use tabs for giving indentations.

Well many of us have the habit of giving the tab by pressing the tab some space is obtained to create the spaces, but the problem with tab is that they are differently handled in different systems. In some of the system a tab may be equivalent to 4 spaces, in some other system it may be equivalent to 6 spaces and so on. So, as you move your code cross machines your structure of your code may look different ok.

And there is another thing here, it says 1 line must not contain more than 1 Verilog statements. Now see earlier in many of the examples, because of the lack of space on the slides. Well I gave some examples like this let us say.

(Refer Slide Time: 15:43)



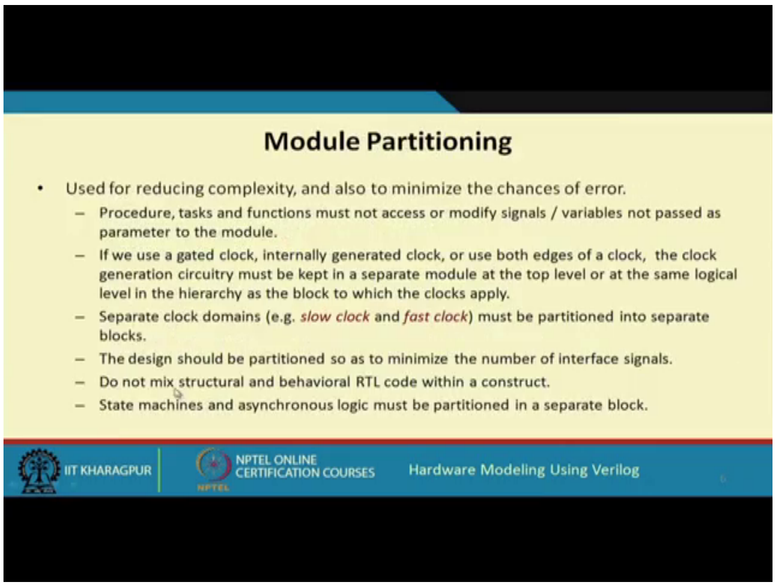
A equal to b plus c x equal to a minus 5. So, 2 statements were given on the same line, but this says that you must not do it this a equal to b plus c should be in 1 line, and x equal to a minus 5 will be in the other line. So, every line must contain 1 verilog statement.

The next 1 is very interesting, it says that well you can give comments in 2 ways; either you can give double slash, which means your entire line is comment or you can use slash star, then comments star slash. This may be a multiline comment, means say in a code if you give a slash, slash in a line then the whole line will become a comment. So, again you give a slash slash, this line will become comment, but if you give a slash followed by a star, then this comment will continue across several lines, then you can give a star slash at the end.

So, what this rule says that to avoid the second kind of comment, you better use this comment, this style right, because at the beginning you will know from the beginning that, well this is a comment line, but here say you are not seeing this you cannot just identify whether this is a comment line or this is part of your code right, that confusion maybe there.

So, just use only a 1 line comment using double slash, and just every line of the code must be within eighty characters, so that the line may not spill to the next line. And again another rule says, whenever you are declaring ports like your declaring a module, whose parameters are a b c typically, you specify a b c on the same line as in the example that I have shown, but this rule says you must specify them on 3 different lines a on 1 line, b on second line, c on third line maybe the size or the number of lines of your code will be increasing, but your structure will be very easily visible. You can see that there are 3 parameters on 3 lines a b and c, it will clearly visible fine.

(Refer Slide Time: 18:14)



Module Partitioning

- Used for reducing complexity, and also to minimize the chances of error.
  - Procedure, tasks and functions must not access or modify signals / variables not passed as parameter to the module.
  - If we use a gated clock, internally generated clock, or use both edges of a clock, the clock generation circuitry must be kept in a separate module at the top level or at the same logical level in the hierarchy as the block to which the clocks apply.
  - Separate clock domains (e.g. *slow clock* and *fast clock*) must be partitioned into separate blocks.
  - The design should be partitioned so as to minimize the number of interface signals.
  - Do not mix structural and behavioral RTL code within a construct.
  - State machines and asynchronous logic must be partitioned in a separate block.

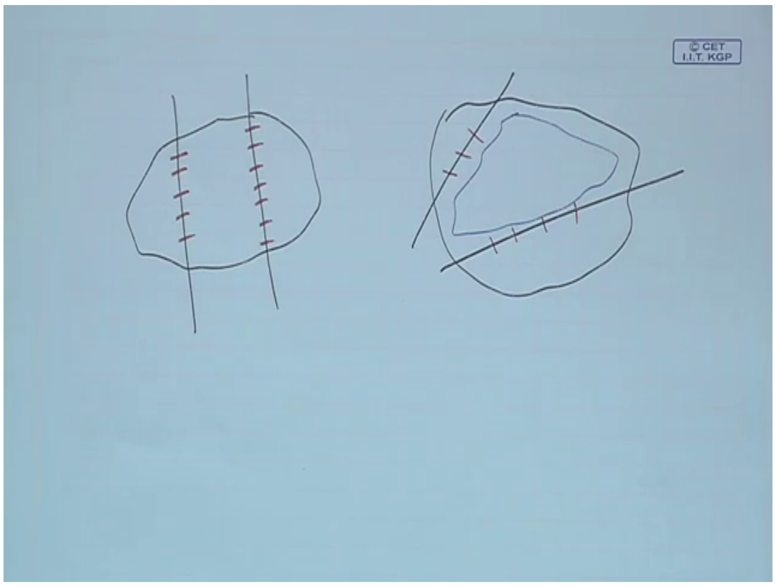
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

Then of course, module partitioning is very quiet obvious. So, usually for complex modules you can divide or split the module into smaller modules, and instantiate them into a top level module; that is normally the design style we form, this is called module partitioning, and this helps in reducing the complexity of design, and because you are handling smaller modules at 1 time the chances of errors will also reduce.

Now, some of the rules regarding module partition says that procedure task functions whatever it is, they must not access or modify signals, not passed as parameter to the module. See verilog allows you to access some variables, even outside the scope of that task or function, but here the rule says you must never do that, you only access variables that are passed to the function, and not other variables. And if you are using some gated clock or internally generated clock, or if you are using multiple edges of the clock, then the clock generation circuitry must be separated out, in a separate module this is what this rule says.

Similarly, if you have multiple clock domains like in a design, there is a slow clock and also a fast clock, they must be defined in separate modules, separate blocks, this is important see given a design.

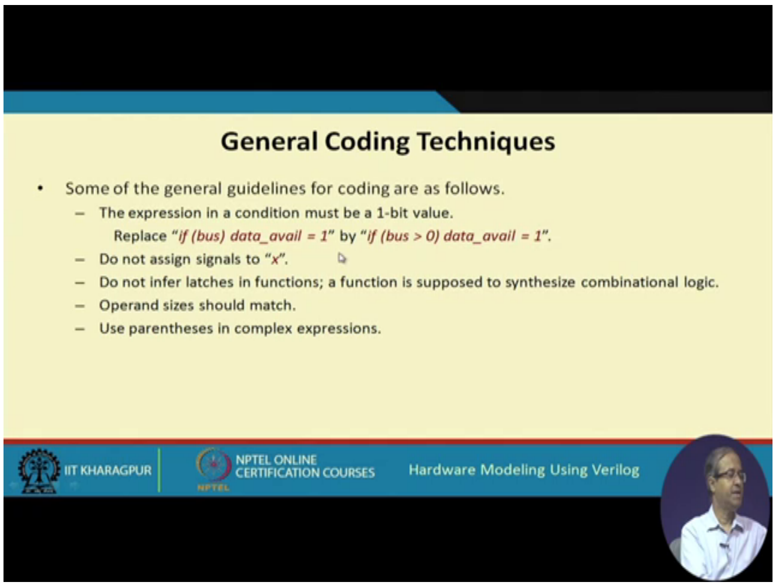
(Refer Slide Time: 20:01)



Say our objective is to partition it to smaller design. Now this same design it is a. I can also partition it like this. Now which of them is better, see you will have to find out how many signals are crossing this partitions, the partition where the number of signals crossing a less that will be considered to be a better partition, because your number of parameters to the modules will be less. So, your design will be much more structured and naturally with this part of your design, will contain most of the self contained code, because they do not have too many interactions with others right. So, this is 1 objective of partitioning a larger design. So, the design should be partitioned.

So, as to minimize the number of interface signals and structural and behavioural code must not be mixed at the same place. Similarly state machines like which rely on a clock and asynchronous logic, which rely on feedback connections and no clock, they must not again be mixed, they must be put in separate blocks or separate modules, right.

(Refer Slide Time: 21:29)



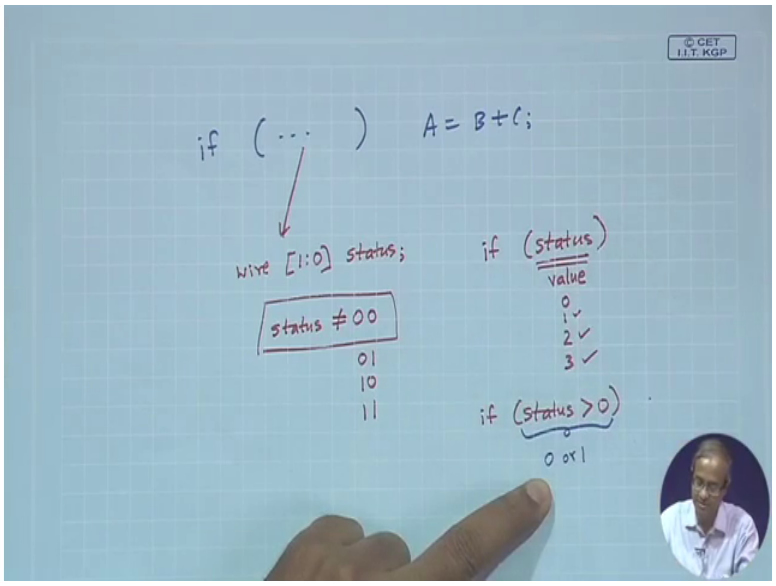
General Coding Techniques

- Some of the general guidelines for coding are as follows.
  - The expression in a condition must be a 1-bit value.
    Replace "*if (bus) data_avail = 1*" by "*if (bus > 0) data_avail = 1*".
  - Do not assign signals to "*x*".
  - Do not infer latches in functions; a function is supposed to synthesize combinational logic.
  - Operand sizes should match.
  - Use parentheses in complex expressions.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

Well, some of the general coding techniques are as follows. Like when you are giving a condition like for example, in the, if then.

(Refer Slide Time: 21:43)



Else statement like if some condition you do something you say a equal to b plus c, but what about this condition. This condition may be the condition that we are checking. let us say you have declared, let us say 2 bit variable, let us call it status. So, here you are trying to check whether status is not 0 0 or not right. So, which means you are checking whether is 0 1 or 1 0 or 1 1.

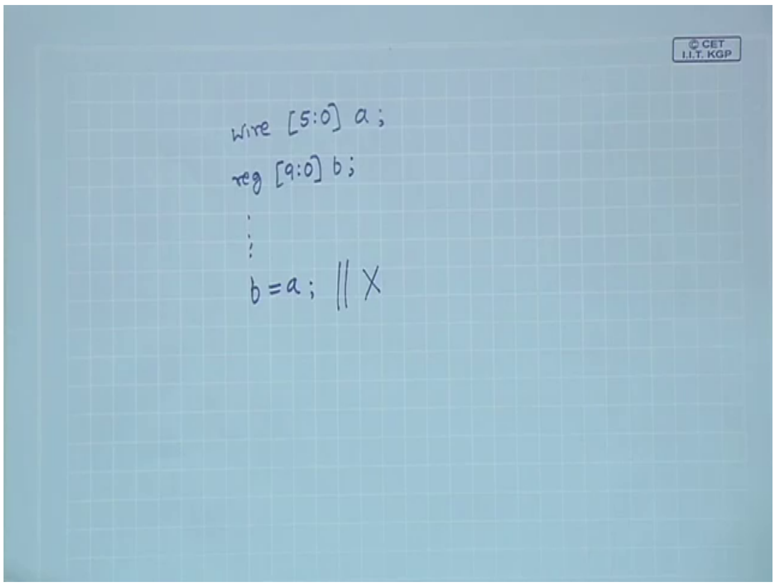
So, you see there are 2 ways in which you can write it. You can either write if simply status. So, what does this mean? Status is a 2 bit number, it will be treated as a number with a value. So, in a 2 bit quantity, the value can be either 0 or 1 or 2 or 3 and any non zero value in the, if condition is treated as true. So, if it is either 1 or 2 or 3, it will be consider as true, zero will mean false, but what it says that you must not use this kind of multi bit values in status checking, rather you use specifically like this.

If status greater than zero, then do this. This will also mean the same thing, but here very specifically saying that status greater than, you see status is a 2 bit variable zero 1 2 3 and greater than is an operator. So, if you consider this expression. This expression can be either true or false, false or true, a single bit value, this expression is a single bit expression, but here status was at 2 bit expression that is why it says the expression in a condition must be a 1 bit value.

So, if bus that same kind of example. So, you avoid this your, rather right bus greater than zero something like this do not assign any signal to x, because you see signals are supposed to be undefined in the beginning, but as computation proceeds you are supposed to initialize them to known values. So, assigning something to x is meaningless. So, you do not do that right. This is of course, clear do not infer latches in functions for multiple branches you specify the conditions completely. So, a function that is supposed to synthesize combinational logic must not generate latches and synthesis.

Here OPERAND sizes should match this is another condition, which is specified here, but of course, Verilog supports unequal OPERAND assignments also like, let us say suppose I have a.

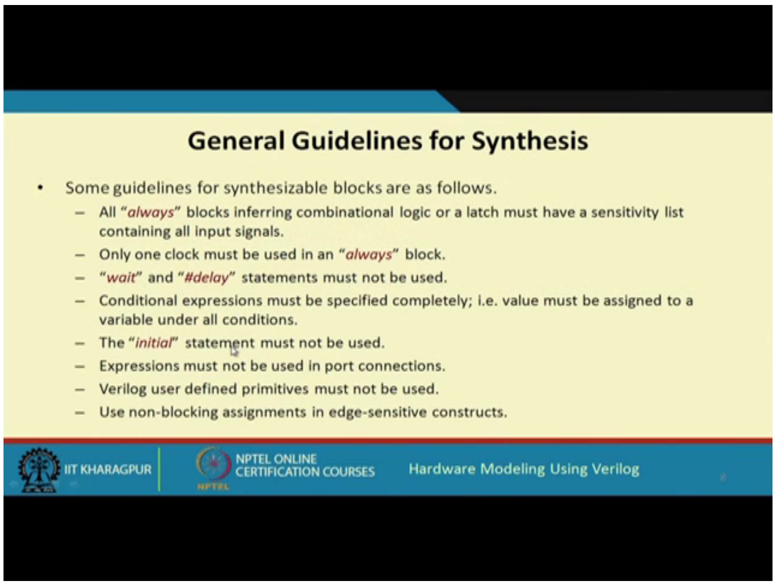
(Refer Slide Time: 25:03)



```
wire [5:0] a;
reg [9:0] b;
   :
   :
b = a; // X
```

Variable, let us say 6 bit variable a and I have another reg type variable, let us say this is 10 bit variable b. So, inside a procedural block I can write b equal to a. So, the Verilog compiler will accept it. So, it will just assign the 6 bit to the last 6 bits, then the higher bits will be set to zero, but this guideline says you must not use this kind of assignment. So, the OPERANDs must all be of the same size. So, OPERAND sizes should match and for complex expressions we use parentheses to clearly specify the presidencies.

(Refer Slide Time: 25:56)



General Guidelines for Synthesis

- Some guidelines for synthesizable blocks are as follows.
  - All "*always*" blocks inferring combinational logic or a latch must have a sensitivity list containing all input signals.
  - Only one clock must be used in an "*always*" block.
  - "*wait*" and "*#delay*" statements must not be used.
  - Conditional expressions must be specified completely; i.e. value must be assigned to a variable under all conditions.
  - The "*initial*" statement must not be used.
  - Expressions must not be used in port connections.
  - Verilog user defined primitives must not be used.
  - Use non-blocking assignments in edge-sensitive constructs.

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

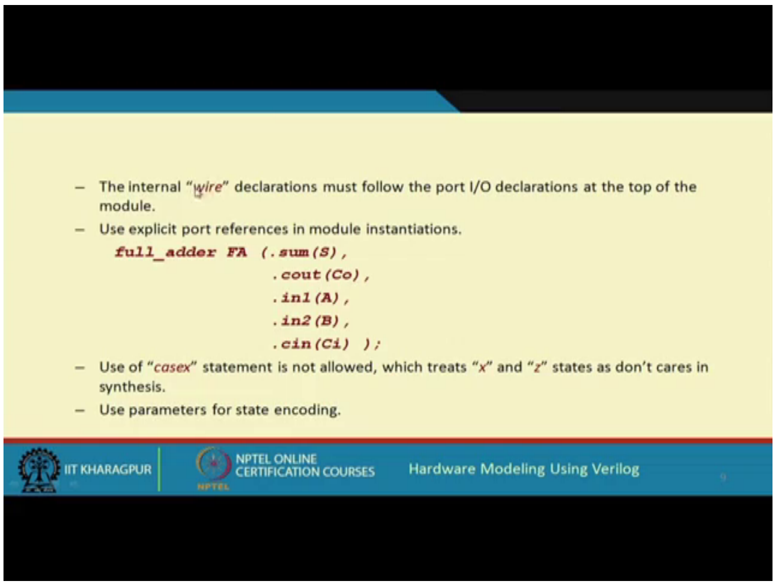
Now, for synthesis already in the last lecture we mentioned a few things, some guidelines have to be followed, like always block which is meant to design a combination circuit. A generator combination logic must have a complete sensitivity list. So, all the inputs to that combinational functional block must be there in this list, and you should not use more than 1 clock inside a single.

Always this wait and delay statements must not be delay, we have already seen delays used to specify time delay for simulation, and wait is a statement like this, wait within brackets some expression this means the expression is evaluated, and it is true or false. If it is false then execution is suspended, whatever statements are after that those will be suspended, means until it is true; that means, this expression will be continually evaluated, whenever it is true then only that block will start executing, but again this is something to do with simulation, not with synthesis, so you should avoid this.

Conditional expressions case if the, if else must be specified completely otherwise some latches might be generated initial statements you cannot use. So, whenever you are instantiating some modules, you must not give expression in terms of ports; like you cannot write in place of some arguments, some x plus y or something like that. You can use only single variables for the parameters or arguments when you are instantiating modules ok.

And this user defined primitives must not be used, even if the synthesis tool might be supporting at least the combination (Refer time: 27:57), but these guidelines says you must not use this, and for h sensitive constructs pauses and (Refer time: 28:05) you use only non blocking assignments.

(Refer Slide Time: 28:20)



- The internal "*wire*" declarations must follow the port I/O declarations at the top of the module.
- Use explicit port references in module instantiations.

```
full_adder FA (.sum(S),
               .cout(Co),
               .in1(A),
               .in2(B),
               .cin(Ci) );
```

- Use of "*casex*" statement is not allowed, which treats "*x*" and "*z*" states as don't cares in synthesis.
- Use parameters for state encoding.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

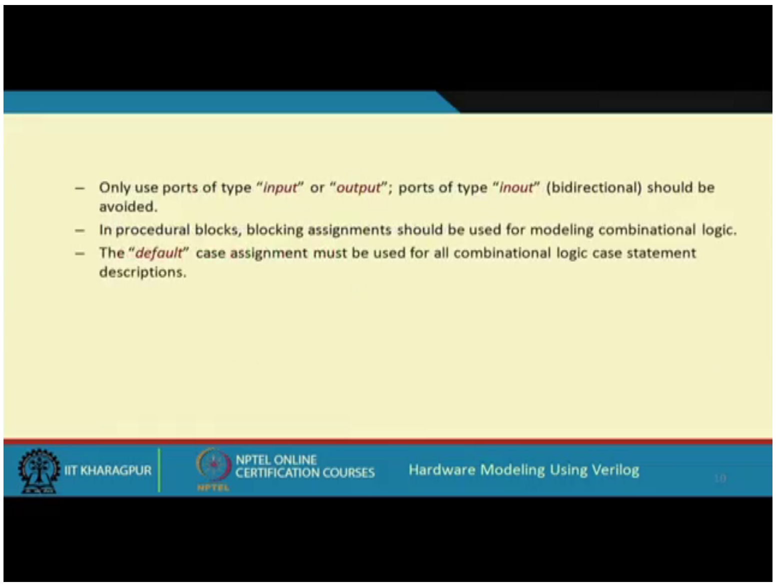
So, the internal wire declarations must follow the I o port declaration which we already do. So, whenever we declare a module, we first define the input and output signals, then we define the wires. So, exactly that is mentioned here that the wire declarations must follow the I o port declarations.

And this is important, it says use explicit port references in module instantiation. So, I said there are 2 ways to instantiate; 1 within bracket to specify the variables in the same order. This is the alternative you mention the exact name of the variables that was present in the module full adder, and this are the variable names in the current module.

The advantage of this notation is that, even if you change the order of these lines, still instantiation will be correct right. So, it is said that you use this style, do not use the other style, and 1 parameter per line like this, and it says you avoid k six, because in k 6 statement this states x and z are treated as do not cares. So, which is not good for synthesis? Synthesis tool might be generating some wrong hardware and for state encoding you use parameters, which we have already seen. We had used the state s zero s

1 s 2, instead of calling them as 0 0 0 0 0 1 0 1 0, it is always means easy to specify something by names rather than by numbers fine.

(Refer Slide Time: 30:07)



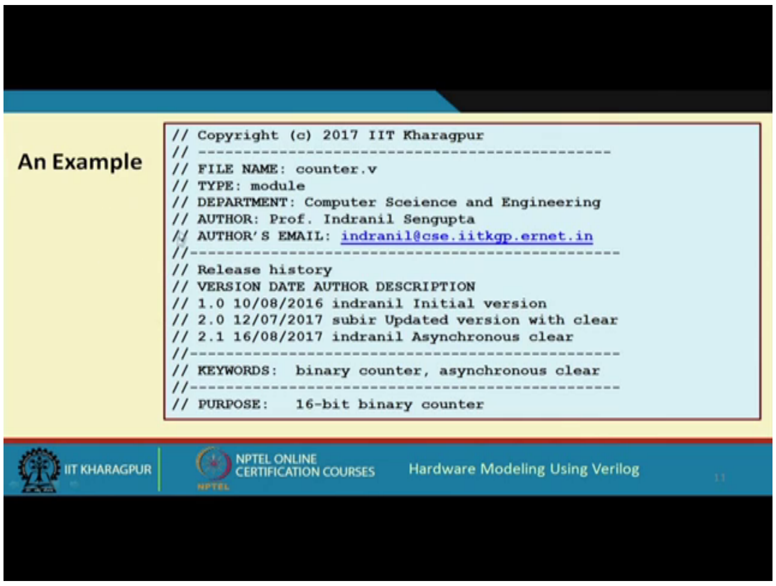
- Only use ports of type "*input*" or "*output*"; ports of type "*inout*" (bidirectional) should be avoided.
- In procedural blocks, blocking assignments should be used for modeling combinational logic.
- The "*default*" case assignment must be used for all combinational logic case statement descriptions.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES Hardware Modeling Using Verilog

And it says we use ports only of input and output type the port inout, which have not discussed in our lectures deliberately, bidirectional port should be avoided. So, the inside procedural blocks, blocking assignment should be used for modeling combinational logic only, and for combinational logic. Again you must use the default statement in case these are some of the guidelines.

So, just a very simple example which shows you the structure of a program, you see the header can be more elaborate. So, this is a very simple example.

(Refer Slide Time: 30:42)



An Example

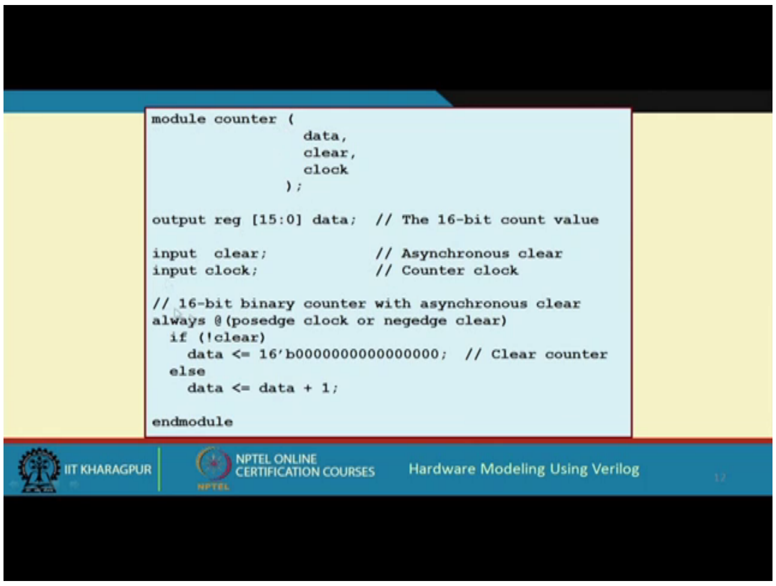
```
// Copyright (c) 2017 IIT Kharagpur
// ---------------------------------------------
// FILE NAME: counter.v
// TYPE: module
// DEPARTMENT: Computer Sceience and Engineering
// AUTHOR: Prof. Indranil Sengupta
// AUTHOR'S EMAIL: indranil@cse.iitkgp.ernet.in
//-----------------------------------------------
// Release history
// VERSION DATE AUTHOR DESCRIPTION
// 1.0 10/08/2016 indranil Initial version
// 2.0 12/07/2017 subir Updated version with clear
// 2.1 16/08/2017 indranil Asynchronous clear
//-----------------------------------------------
// KEYWORDS:  binary counter, asynchronous clear
//-----------------------------------------------
// PURPOSE:   16-bit binary counter
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

You see there are so many lines were using for the header copyright, some copyright information or message may be there, then file name, what is the name of this file, what is this type, is a module department. Authors email, because if there is some query you can contact the author, release history. There are multiple versions of these. So, all the details must be there, what was the version name, which dates the version is created, who was the author and what was the modification. Some of the name keywords and some purpose description here I have given, it in a very brief way sixteen bit boundary counter.

You see indentation.

(Refer Slide Time: 31:34)



```
module counter (
                  data,
                  clear,
                  clock
                );

output reg [15:0] data;  // The 16-bit count value

input  clear;            // Asynchronous clear
input clock;             // Counter clock

// 16-bit binary counter with asynchronous clear
always @(posedge clock or negedge clear)
  if (!clear)
    data <= 16'b0000000000000000;  // Clear counter
  else
    data <= data + 1;

endmodule
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
Hardware Modeling Using Verilog

This is the module description where instead of putting them on the same line, we have specified them on separate lines, this is the output, these are inputs and with all the variable parameter declaration, there is a comment which specifies the purpose of that parameter, then with every block some explanation is there right, and here also wherever relevant we had comments clear counter.

So, this is just an example. So, with this, we actually come to the end of this lecture. Actually the example that I have showed you, it is just for you to have a glimpse of the feeling. So, you may think that. Well my task is to write the module, to create the design. So, why I shall waste my time in writing the header creating the comments, but well this is extremely important, if you write a correct module without any comments it is as good as useless. No 1 will be using your module in the future, because in the industry mainly the main emphasis today is on design, re-use.

Suppose, I design some complex subsystem today, tomorrow someone else may be trying to use my design to create a more complex design. So, that person will be trying to re-use my design. So, unless my design is very well documented, this will not be possible.

So, with this we come to the end of this lecture.

Thank you.