

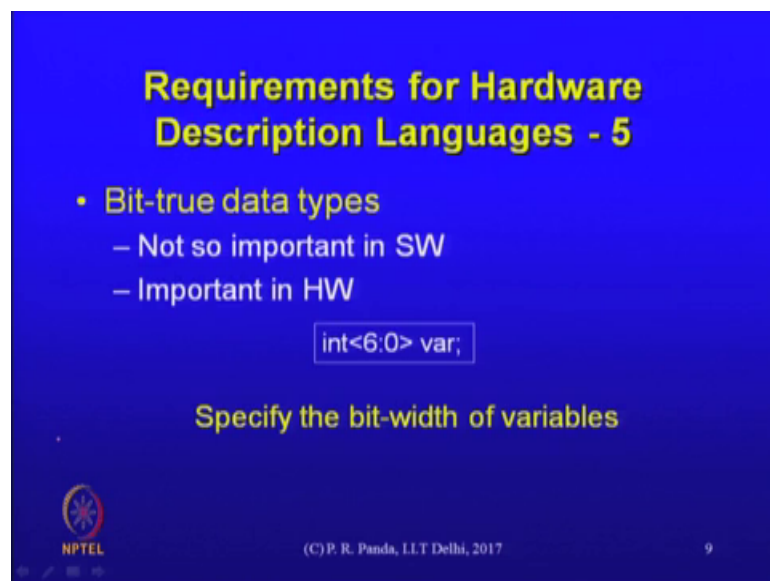
Synthesis of Digital Systems
Dr. Preeti Ranjan Panda
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi

Lecture – 03

VHDL: Introduction to Hardware Description Languages & VHDL Basics

So, let us continue with this requirement, a set of requirements we are talking about regarding specifying exact bit widths and we argue that it is its essential.

(Refer Slide Time: 00:17)



Requirements for Hardware Description Languages - 5

- Bit-true data types
 - Not so important in SW
 - Important in HW

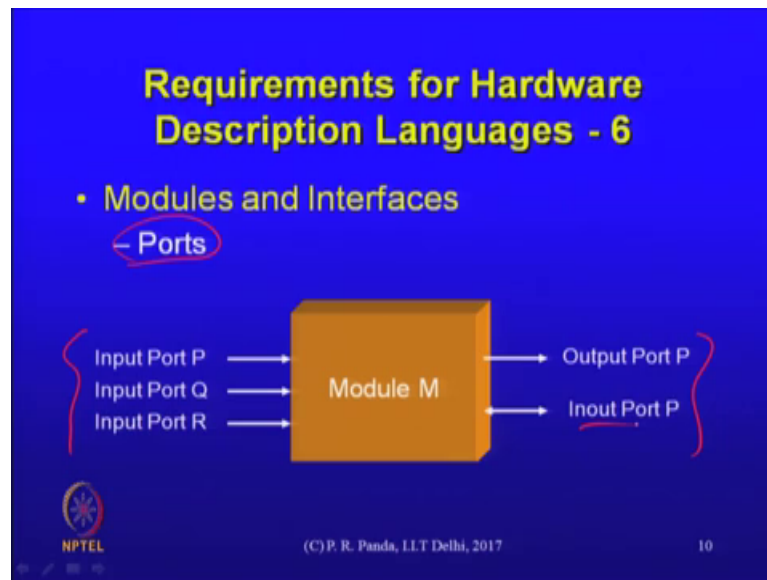
```
int<6:0> var;
```

Specify the bit-width of variables

NPTEL (C) P. R. Panda, IIT Delhi, 2017 9

Let us move on to some other elementary requirements for HDLs.

(Refer Slide Time: 00:35)



The idea of dividing a design into smaller modules and connecting those modules up in some way is an elementary requirement in hardware design and therefore, its specification. So, I should be able to create a module this terms that I have picked up here module and so on these are some somewhat generic terms they are not language specific really.

What is the first thing that comes to mind when I design a module, what is the external interface of that module? External interface is essentially defined through a set of ports here what we have here is a set of input ports and a set of output ports maybe bi directional port if there is one. What are the set of ports I should be able to define in the module? That constitutes certain elementary level the external interface of the module.

(Refer Slide Time: 01:36)

Requirements for Hardware Description Languages - 7

- **Electrical Characteristics**
 - Current Levels
 - Tristating
- **Sensitivity**
 - rising edge/falling edge

NPTEL (C) P. R. Panda, IIT Delhi, 2017 11

What other things might be basic requirements? Some electrical characteristics like current levels and so on are optionally relevant here that does depend some of these depends on the kind of technology that your system electronic system is based on, but the idea of tristating something is nowadays considered as sort of an elementary requirement.

So, a language might provide some way of specifying that a particular output of a buffer is tristated itself, as we know that helps in designing certain kinds of electronic systems. Sensitivity to a clock edge is an elementary requirement in the specification of a digital system. You would like to say on the rising edge of a clock certain thing is triggered right. So, that is a sensitivity rising edge or the falling edge that is a requirement. So, let us include that also among the elementary requirements in HDLs.

(Refer Slide Time: 02:38)

Requirements for Hardware Description Languages - 8

- Other programming constructs
 - Text and File I/O
 - useful in simulation/debugging

NPTEL (C) P. R. Panda, IIT Delhi, 2017 12

I could list a number of these, but one orthogonal requirement is what about some programming constructs like file IO string processing and so on. Is that useful? It is. Why is it useful?

Student: (Refer Time: 02:53).

Remember it may not be directly useful in specifying our design itself right, but there may be other uses of this file IO which you can argue its useful for design or not, but certainly from an overall design environment it is essential because its useful in debugging and so on, while the simulation is going on I may want to print out something.

So, some such constructs file IO related constructs, string processing related constructs are usually included in HDLs right, even though they are not necessarily relevant for specification of the design itself. Design is not the only thing that we are doing we are also doing lots of other things we are generating inputs for testing right that is not part of the design if you are generating inputs for testing, but how do you generate that input. It is considered so basic that the design language itself provides some mechanism for you to be able to specify the input itself right or equivalently if we just extend that you should be able to specify your input in a file and in your HDL you read that file and provide that as an input to the design right. So, those things are part of an overall design environment and modern HDLs do provide some mechanisms for that.

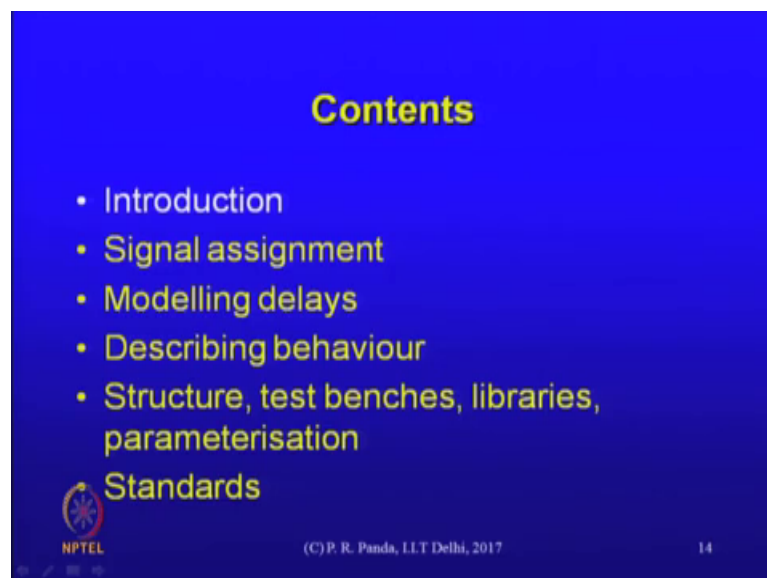
So, these are some simple requirements.

(Refer Slide Time: 04:20)



Let us move on to a specific HDL and illustrate a number of these features in the context of VHDL.

(Refer Slide Time: 04:29)

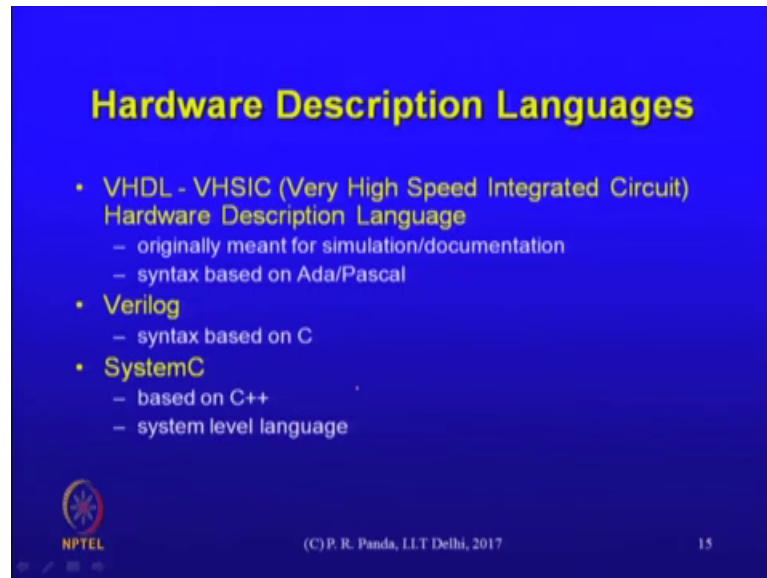


Student: (Refer Time: 04:29) where do we use some current levels in the HDL.

Where do we use current levels in the HDLs? We will point out examples in the VHDL context where I am able to specify the strength of a signal that is indirectly specifying the


current level. It is useful in the various contexts as you know it is represented differently in different languages, but in VHDL how do we do that we will get to. So, just hold on ok.

(Refer Slide Time: 05:02)



Hardware Description Languages

- **VHDL - VHSIC (Very High Speed Integrated Circuit) Hardware Description Language**
 - originally meant for simulation/documentation
 - syntax based on Ada/Pascal
- **Verilog**
 - syntax based on C
- **SystemC**
 - based on C++
 - system level language

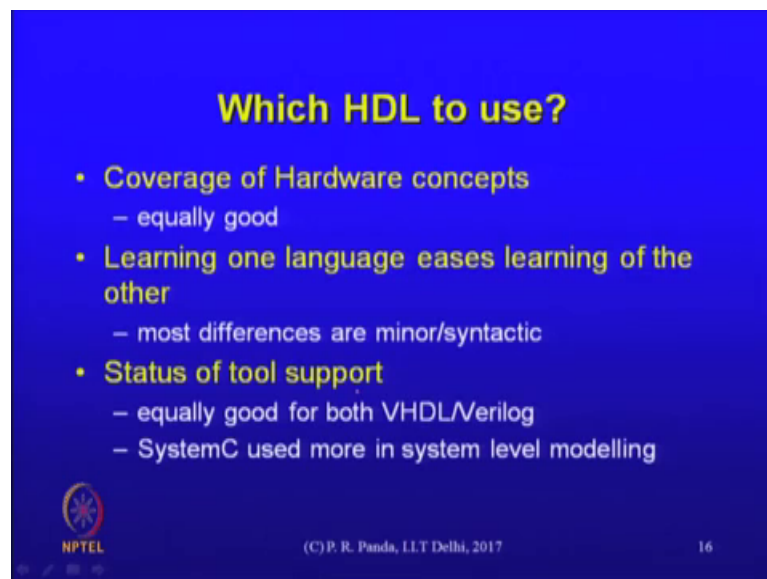
 NPTEL (C) P. R. Panda, IIT Delhi, 2017 15

So, this VHDL is a recursively defined acronym that V is an acronym by itself it stands for very high speed as C. This language was originally meant for simulation and documentation and so on when you deliver a design there is the need for a formal documentation and there was a specification language and that is what it started off as that syntax is based on an earlier generation of programming languages, Pascal and Ada. Interestingly at that point nobody was thinking that this specification in language like VHDL or other languages like verilog are necessarily meant as a starting point of a design. It is more like an afterthought to formally specify what the design is doing after you have designed it that is how it started off, now the methodology is different but you have built a design out of whatever schematic editors and so on.

And now you want to capture it. So, that it could be simulated somebody who is buying the product can simulate it in his environment, but nobody was thinking that this could actually be treated as the design itself and we start our design methodology our design process from the language. But this is from long ago when the language was first designed, since then people found value in just treating that itself as the entry point for the design.

The syntax is slightly different for the very log the syntax is sort of based on C, the third language I have put up there is system C it is based on C++ positioned as a system level language. So, we talked about what that system level of abstraction was. It is the level of abstraction that captures the design intent before you have necessarily made up your implementation decisions of whether something will be done in hardware or in software and so on, but hardware constructs of the kind that we look for in HDLs are also present in system C.

(Refer Slide Time: 07:04)



Which HDL to use?

- Coverage of Hardware concepts
 - equally good
- Learning one language eases learning of the other
 - most differences are minor/syntactic
- Status of tool support
 - equally good for both VHDL/Verilog
 - SystemC used more in system level modelling

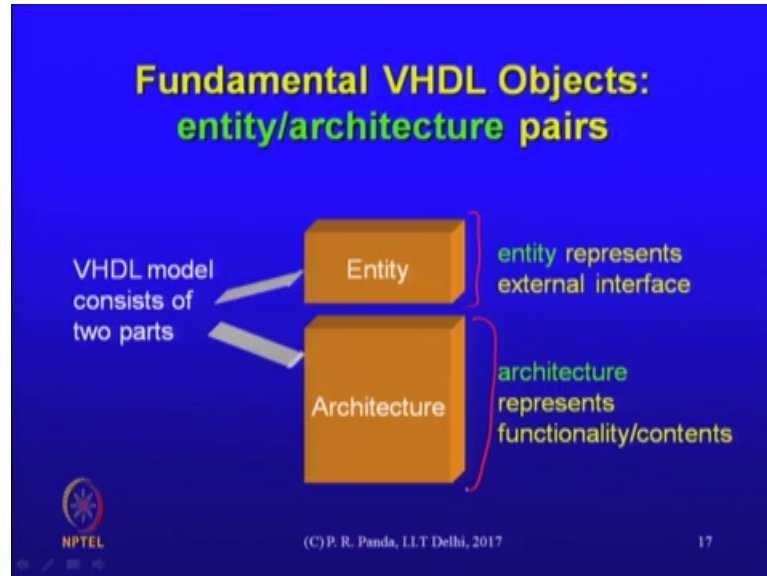
NPTEL (C) P. R. Panda, IIT Delhi, 2017 16

The question of which one to use is sort of unclear it is not very important, we will try to in this course point out things that are of generic importance in HDLs. We will use a specific example of a language, but that is just an example most of the ideas, important ideas and span the multiple languages they are present in all the languages. So, of course, learning one language eases the learning of the other and most of the differences are sort of minor and only syntactic variations. Not all, but most of them are common.

As far as tool support is concerned they are equally good for both VHDL and verilog system C like I said is used more in system level modeling it does provide support for a lot of the hardware constructs that we talked about also, but it is not necessarily particularly useful additionally over the number of features that are there for hardware description might not be too much more than what is already there in HDL. So, anyway that is why while it is a popular language for modeling systems at a little higher level of

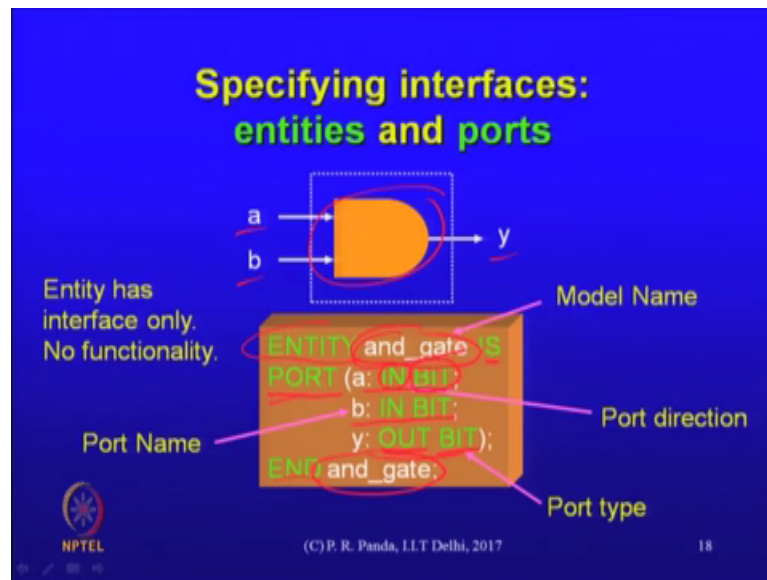
abstraction, the advantages are not necessarily there if you are going to describe a plain digital hardware system with it.

(Refer Slide Time: 08:19)



So, with that let us move to some discussion about VHDL. At a high level there is an entity architecture pair in a language like VHDL, that entity essentially represents an external interface of the design. We talked about the requirement for an HDL to support specification of modules and their interfaces, so that is what this is entity is the external interface the architecture is what captures the functionality right, how the design works is what is captured in the architecture part. The entity part just has the external interface. If you were to use that module what is visible to you, how you would connect your signals to this module that is the information that is there in the entity it is the architecture that contains the implementation of that module.

(Refer Slide Time: 09:15)



So, let us just move on to the specification through some examples. I have used a little low level of abstraction small gates in order to illustrate these concepts that is not to say that you can only describe small gates using these languages, you can very well describe very complex design so, but this is there just to illustrate some of the basics syntactical aspects.

So, I have a gate level design a NAND gate here that I would like to model what is the external interface. So, even before I specify the functionality I need to specify the external interface. What is the external interface of a NAND gate? I have two input bits. So, it is like single bit inputs this is a two input and gate, and there is one output that is all that is the external interface and that I need to capture that in an entity specification. So, I create a list of ports. So, each of these the abc each is a port of the entity and a port list is specified that is all that is there in the entity. So, this here is a name that we have given to the model right, we can choose whatever name it is. What you see in green here are all the key words of the language. So, entity means something is needs to be there as part of the syntax the port means something and you conclude this port list with an end statement, and that name there has to match the name of the entity. That is just the elementary syntactic requirement.

Look at the port list I have a list of port specifications right something for a something for b and something for y. What information are we giving? Along with a which is a

name that we have chosen I have two other things one is the direction saying it is an input port the other is the type right I am saying that this is a bit type, that is why this is a two input gate and each input is one bit wide. So, that needs to be specified if it is a more complex is like an integer I need to specify that it is an integer. The output similarly is captured with that out keyword I say that is the direction and that bit type is the output that is all, that is there as an external interface of the AND gate. Remember we did not yet say what it is doing we are just saying that this is the external interface, yeah.

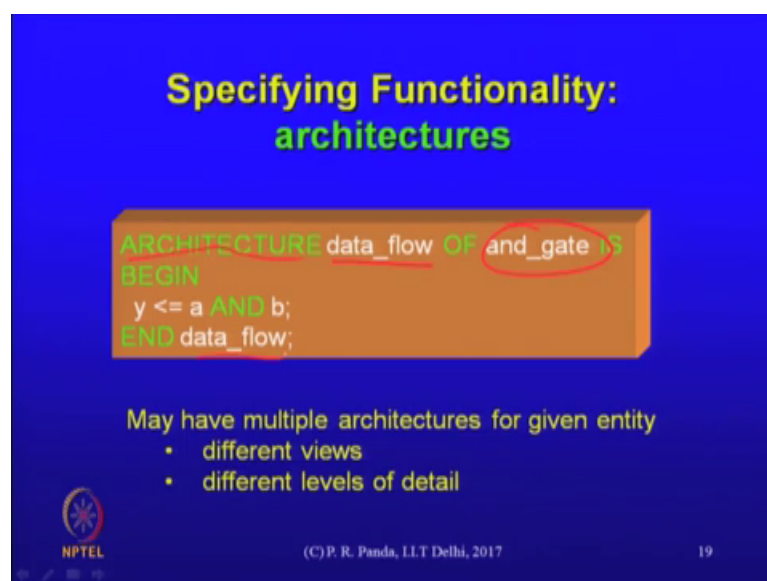
Student: Sir, that and gate the model name (Refer Time: 11:45).

Yeah. So, what you see in white here write the model name this is our choice that is only what is there in green is predefined, this is part of the language syntax. We could have chosen anything else, we could have chose instead of aby we could have chosen any other names instead of the AND gate we could have given something else.

Student: (Refer Time: 12:04) colons and semicolons.

The colons and semicolons are part of the language, yeah. Any question regarding the external interface, this is simple enough right. Of course, I can specify any number of ports as necessary and while these are the elementary bit types I can have other types for specification.

(Refer Slide Time: 12:28)




**Specifying Functionality:
architectures**

```
ARCHITECTURE data_flow OF and_gate IS
BEGIN
  y <= a AND b;
END data_flow;
```

May have multiple architectures for given entity

- different views
- different levels of detail

 NPTEL (C) P. R. Panda, IIT Delhi, 2017 19

Let us move on to the other part the architecture part of the design that is where we specify what it is doing remember, or how it works. So, the architecture is a key word that is where I specify the functionality. The AND gate here tells us that this is the connection between the entity specification and the architecture which entity are we specifying the implementation of that connection is established by using that gate, using that name that we gave to the entity.

This is a name of the architecture, we are saying we chosen a name called data flow here it does not really matter you can choose a name of your choice. If it is descriptive in some ways it is good, but the language provides a way for you to specify multiple architectures for the same entity. Why would that be useful? You could put it to many different uses, but you could for example, capture different levels of detail of that same design as time progresses maybe you start off with a very approximate design it just represents, so many maybe a very high level algorithmic description timing is not there and so on right. It is still useful to model like we said at a high level of abstraction, but later on you may choose to go for a more detailed implementation that could be captured in a different architecture of the same entity. The entity is not changing the external interface is not changing only the implementation is changing. So, this could be a way it for us to capture different implementations of the same module.

Ultimately when we take it up later for simulation or synthesis or something we have to indicate which architecture, which implementation we would like to implement we would like to synthesize or we would like to simulate at this stage, but the language provides us the ability to specify different implementations that would be by specification of different architectures yeah.

Student: (Refer Time: 14:26).

It is just the name that we have given why we have chosen this maybe we will get to later, but you could choose something else.

Student: (Refer Time: 14:33) for example, if we have to define multiple architectures for AND gate. So, we can give them.

We give them different names.

Student: (Refer Time: 14:40) and then we have to use them it will specify the entity name.

We specify, yeah when we use it we specify both the name of the entity and the architecture of the entity yeah.

Student: During instantiation (Refer Time: 14:50).

During instantiation, but we may be running ahead because, yeah that is a use an example of a use of the entities.

So, that is just the interface specification we did not exactly explain that functionality there. Functionality is captured in that one statement, but that is what we are coming to.

(Refer Slide Time: 15:12)

The slide features a blue background with yellow text for the title: "Specifying Concurrency: Concurrent Signal Assignment". On the left, a brown box contains VHDL code for a full adder architecture. The code is:

```
ARCHITECTURE data_flow OF full_adder IS BEGIN si <= ai XOR bi XOR ci; co <= (ai AND bi) OR (bi AND ci) OR (ai AND ci); END data_flow;
```

 Red circles highlight the signal assignment statements, and a red bracket groups them under the label "Concurrent Signal Assignments". To the right, a logic diagram shows a full adder block with a plus sign. It has three inputs: ai, bi, and ci. It has two outputs: si and co. A red bracket on the right side of the diagram groups the outputs.

Let us start with the specification of the functionality. At an elementary level the feature that one would use to specify functionality is what is called a signal assignment. We will define what the signal is, but as of now we have an entity this is full adder structure which has what inputs. It has three inputs there is a b are the data inputs, a ci is a carry input and si is the sum and co is the carryout, so three inputs and two outputs from the adder. I did not show the entity here, but you can easily see how you would represent the entity from that structural information.

So, full adder is the name of the entity that is the name of the architect it a flow is the name of the architecture we have chosen and between the begin and end of the architecture we have to specify the functionality all the functionality has to go there. What is our functionality? I am saying that the sum si is assigned the value of ai XOR bi XOR ci this is the standard specification of the functionality of the some bit of a full adder. That operator there is the signal assignment operator there is a co which is the carry out carry out has a different logic it has the a b plus bc plus ca logic. So, that is what is captured in that other statement. So, I have two signal assignment statements> What is important to note here which is a language specific feature is that these are concurrent signal assignment, concurrent means that the order in which you specify them is not important both of them are active at the same time.

(Refer Slide Time: 17:17)

When is Signal Assignment Executed?

Assignment executed when any signal on RHS changes

```
ARCHITECTURE data_flow
OF full_adder IS
BEGIN
  si <= ai XOR bi XOR ci;
  co <= (ai AND bi) OR (bi AND ci)
        OR (ai AND ci);
END data_flow;
```

Executed when ai, bi, or ci changes

Executed when ai, bi, or ci changes

NPTEL (C) P. R. Panda, IIT Delhi, 2017 22

So I need to little carefully understand what do you mean by both statements are active at the same time to answer that we have to answer this question of when is a signal assignment triggered or executed. You have written a bunch of signal assignments here that is the specification of the functionality of the full adder the rule is the assignment is executed when any signal on the right hand side of that assignment changes right, I have that expression there that expression consists of this ports as of now they are ai bi ci all are ports, si is also a port but these are the input ports right ai bi and ci.

So, the rule is that that assignment is triggered that particular assignment statement is triggered possibly resulting in a new value to s_i , if you have changes in a_i or b_i or c_i . If any of those signals changes then the output is evaluated afresh. Now, remember that may or may not lead to a change in the output that depends on what is the logic that you have given, but the expression is evaluated once more that is the rule. And of course, if that the second statement is also a function of a_i , b_i and c_i right. So, if any one of those three input signals changes both in this example here both the statements would be evaluated. Any basic questions on the rule? The rule is simple is it intuitive does it make sense to have this rule.

Student: (Refer Time: 18:48).

Conversely, what it means is that if those signals did not change if any of those three signals did not change then the evaluation of s_i would not happen, would not happen means it somehow retains the old value that was computed, yeah.

Student: Sir, but that does not mean that the signal assignment of s_i and c_o has to be concurrent (Refer Time: 19:10).

Yeah, by concurrent we will define in a little closer detail what exactly it means. We are saying that these are independent of each other right. In this particular case the both are a function of the same three variable there may be functions of other variables of course. If they are then the triggering of the first statement is independent of the triggering of the second statement. But does it make sense to have a rule like this?

Student: Yes.

The consequence of this is that if they did not change if those inputs did not change then we will not reevaluate the output, will not reevaluate means the output retains whatever was the last computed value for the output. Does not make sense?

Student: (Refer Time: 19:52).

Why does it make sense?

Student: (Refer Time: 19:54).

No, no function, but is it correct first of all according to the way we think about the hardware design.

Student: Yes.

It is correct, in CMOS logic it is correct, right there is the dependence there on what technology you are using may or may not depending on what kind of gates what kind of transistors you are using, but that logic family works in a way that if the input is not changing then the output is maintained right. If that were not the case then this would not be the rule it would be the something else.

But otherwise the rule makes sense that is what is there, if you have a number of signal assignment statements all of them are subject to the same rule which is right hand side and left hand side should be clear from this, on left hand side we have the signal that is being assigned here it is the port that is being assigned. On the right hand side I have all the signals that it is a function of, so it intuitively makes sense that if anything changes then the output may change right.

(Refer Slide Time: 20:54)

Order of Execution

- Execution independent of specification order

```
ARCHITECTURE data_flow
OF full_adder IS
BEGIN
  si <= ai XOR bi XOR ci;
  co <= (ai AND bi) OR (bi AND ci)
        OR (ai AND ci);
END data_flow;
```

```
ARCHITECTURE data_flow
OF full_adder IS
BEGIN
  co <= (ai AND bi) OR (bi AND ci)
        OR (ai AND ci);
  si <= ai XOR bi XOR ci;
END data_flow;
```

These two are equivalent

NPTEL (C) P. R. Panda, IIT Delhi, 2017 23

Concurrent signal assignment also means that the execution order is independent of the specification order. So, I wrote these two statements here first this and then this. If I had instead interchange order of the two statements right that would still be an equivalent design, that is the rule of the concurrent signal assign. Concurrency means that they are

decoupled there is an elementary rule that is followed for every statement, but each is treated independently and no particular importance is given to the order in which we are writing these statements.

This is as opposed to a sequential programming language in which if you have one statement and you have another statement particularly if there is a dependency between those two statements then the order is important. If you interchange the order you may get a different result. There by here if you interchange the order of specification then you will not get a different design. How do you establish that we will get to, but this is the rule of the language. A simulator that simulates your design has to obey that rule and of course, we as designers need to know that rule also.

Student: If you are writing sequential algorithms.

We will get to how to write sequential algorithms. As of now this is only a matter of concurrent signal assignment specification. There I do not have the choice I need some other construct within the language to actually specify a sequential algorithm, yeah.

(Refer Slide Time: 22:32)

Modelling Combinational Logic

- One concurrent assignment for each output

The slide features a diagram of a 'Comb Logic' block with four inputs (i1, i2, i3, i4) and four outputs (o1, o2, o3, o4). Red circles highlight the input and output labels. To the right, a code block shows the VHDL architecture for 'data_flow' of 'comb_logic' with concurrent assignments for each output: o1 <= i1 and i2; o2 <= (i2 or i3) xor (i1 and i4); o3 <= ...; o4 <= ...; The code is enclosed in BEGIN and END statements.

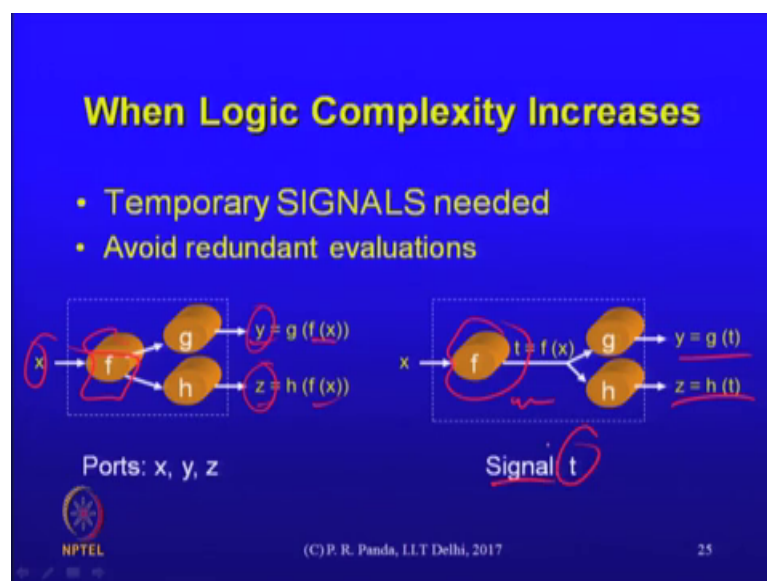
NPTEL (C) P. R. Panda, IIT Delhi, 2017 24

This is good enough to model any combinational logic right. So, combinational logic in general I have a bunch of inputs, a bunch of outputs and how do I specify that I need to model this combinational logic there should be an easy way for me. Basically I have one

statement for every output right for this output. I have a different statement each output results in a different statement.

In here I would capture on the right hand side whatever function it is of the inputs. Of course, ultimately it is a function of all the inputs all the outputs are functions of the input. So, what that respective function is I would capture on the right hand side of the signal assignments and on the left hand side I would just have the output ports here. Hopefully this is obvious.

(Refer Slide Time: 23:26)



There is one simplification we may need to perform which is that I have a system in which there is one input and I have multiple outputs, but the logic may be complex right. So, it may be a very complicated functionality and not just complex in fact, the computation for y and z might actually have some elements that are common right, that is what we have represented here it is some function of f, so I have a function of f is a function of x, and g and h are two different functions.

But that f part is common between both y and z. So, when that happens if I restrict myself to having just this syntax of independently specified concurrent signal assignments then you see that I have to repeat that f part of it the common part of it in both the statements right that is the only way to get around that what I will need is a way to specify first that this is the common computation and let us say I create an intermediate signal object called t and then y and z can then be specified in terms of that

common object. So, that would be a way for me to avoid the duplication of the f that comes in both the expressions. If that expression is complex then it makes sense to do this. So, that is the introduction of a signal.

This is a new syntactical structure beyond the ports that are there. So, you see that signal corresponds to something that is internal to the module. Ports correspond to the external interfaces. So, this signal is useful only for the internal computation the outside world does not get to see the t at all. That is how we would like to structure this.

(Refer Slide Time: 25:16)

SIGNALS

- Represent intermediate wires/storage
- Internal - not visible outside entity

```
ENTITY comb_logic IS
PORT (i1, i2, i3, i4 IN BIT;
      o1, o2 OUT BIT);
END comb_logic;

ARCHITECTURE data_flow
OF comb_logic IS
BEGIN
o1 <= (i1 and i2 and i3) xor i2;
o2 <= (i1 and i2 and i3) or i4;
END data_flow;
```

```
ENTITY comb_logic IS
PORT (i1, i2, i3, i4 IN BIT;
      o1, o2 OUT BIT);
END comb_logic;

ARCHITECTURE data_flow1
OF comb_logic IS
SIGNAL temp: BIT;
BEGIN
temp <= (i1 and i2 and i3);
o1 <= temp xor i2;
o2 <= temp or i4;
END data_flow;
```

NPTEL (C) P. R. Panda, IIT Delhi, 2017 26

Syntactically here is what we would do. This on the left is the original entity and the architecture specification and you see that there is one part of it that is common that i 1 and i 2 and i 3 that is common to both these statements right. So, if that expression is complex then the argument is that we should not need to specify twice what I will do is we will take that common expression out and we create, a new assignment statement now there are three assignment statements three concurrent signal assignments in our design.

I create a new signal called temp of the same type that is also a bit type that intermediate computation is assigned to temp and that temp now features in that other two expressions. You can see that this is a standard way of simplifying the overall complexity of the specifications. This should be nothing new we do it all the time in programming languages of course, when there is a common computation we take it out and form some

other structure with it and use the result in different places that is the same thing here. Remember that the temp was supposed to be an internal signal internal object to this module and should not be visible outside. That is accomplished through declaring that temp as a signal here within the architecture. So, the external world only sees the entity does not see what is outside, so it does not get to see temp at all we get to see temp inside, yeah.

Student: Since we can represent intermediate wires oblique storage.

Yeah.

Student: So, what is the mean intermediate wire?

As of now this is just the functionality, but of course, later on I would need to implement this functionality in some design. What would it translate to if I have a temporary signal that I have assigned? If course, what the ports correspond to in our real design once it is there should be clear, they just represent the external interface they are pins of the design. But if I have an intermediate signal like that what would that correspond to. As we talk about the specification of the hardware it is a good idea since we are already familiar with the hardware design what would the corresponding design be, right.

Student: (Refer Time: 27:34).

Ok. So, now, if I say that this is a temp signal. So, every construct that we have here we should be thinking about what is the equivalent hardware. Remember in that is the whole synthesis process that I should be able to take from this language syntactical specification and generate for you the design. So, you better have an idea even before we talk about synthesis then what to expect from a hardware specification like this.

So, this temp should result in what kind of hardware?

Student: Storage.

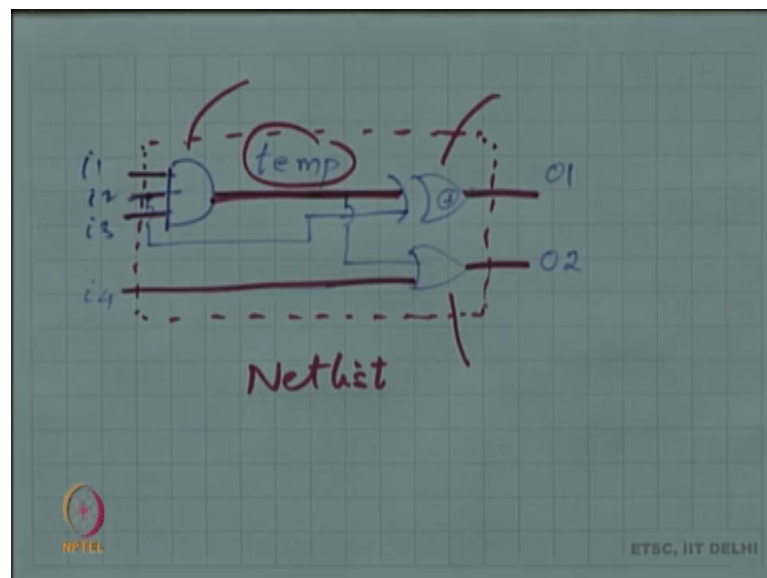
It could be storage, it could be.

Student: (Refer Time: 28:10).

We are just wire it could be just an interconnection, that does depend on what you are doing with it and you would infer some storage if it is really essential if it is not essential then it could just be a wire and so on. This is specification of pure combinational logic. Even this two is combinational logic right because both are combinational functions ultimately which means that ultimately they would be either gates or wires you do not need any storage in the design. So, this, intermediate signal can you imagine what the hardware would be for this structure. Here this is just a three input AND gate right. The functionality part of it is three input AND gate and the output of that AND gate, so our design would be i_1 and i_2 and i_3 that is my temp. What else do I have here?

Student: (Refer Time: 29:13).

(Refer Slide Time: 28:58)



The temp actually leads to two different gates here one is an XOR gate where the second input is i_2 ; that means, I take it from here and the output is O_1 . What else am I doing here? I have an OR gate whose output is O_2 and that also takes temp and the other input here is i_4 , that is the combinational logic.

It should be obvious as of now, the implementation is obvious. So, hopefully we get to more complex specifications of HDL where the implementation is not obvious. That is when it makes to introduce synthesis and automation and so on, but this for the small example it should be clear. And what we have here is that this specification then of temp if you see all the other connections here all the other wires that I have in fact, are part of

the external interface of the design right. The gates are of course, internal, but other than that the all other wires or all other pins they these wires directly lead to the pins and they are part of the external interface of the design. So, it is just this that is internal and that is inferred from our internal signal that we had in the architecture specification.

Student: Sir, HDL compilers in (Refer Time: 31:12) to overcome design stage and find temporary signals on their own, if in case we have (Refer Time: 31:20) designer has not.

Our HDL compilers smart enough to infer mistakes of the user of the kind that the temp was not declared and suggested that there should be a temp signal. You used temp, but you did not declare it.

Student: Sir, what I was meant was in the RTL sorry the HDL on left and right can be interpreted into two different ways.

Yeah.

Student: When you actually put a structural netlist.

Right.

Student: You can duplicate the two and operations there and clear to three input and gates.

Yeah, yes. So, I have some redundancy in the original specification that I have manually optimized in my in that second specification. Your question is are the synthesis tools smart enough to see that themselves instead of asking us to manually identify that something is common here it was obvious sometimes it might not be that obvious. Remember you might have specified it in some another set of operations it might still be a three input AND gate right. Are the synthesis tools clever enough to detect that themselves? They absolutely are. Yeah of course, there is a limit to that smartness, but that is indeed the topic of the later stage the logic synthesis when we study the idea is absolutely to optimize the logic that is generated. There is a straightforward logic that can be inferred from the way the user wrote the code, but that is only the starting point.

The whole point of logic synthesis and all the synthesis and optimizations is that you generate the output that is an optimized netlist that need not have any structural

resemblance to what was specified by the user it should have a functional equivalence, but it need not have a structural resemblance yeah. A simpler redundancy like this would absolutely be identified by a modern synthesis tool.

(Refer Slide Time: 33:30)

SIGNALS

- executed when i1, i2, or i3 changes
- executed when temp or i2 changes
- SIGNALS are associated with time/waveforms
- PORT is a special type of SIGNAL

```
ARCHITECTURE data_flow
OF comb_logic IS
SIGNAL temp: BIT;
BEGIN
temp <= (i1 and i2 and i3);
o1 <= temp xor i2;
o2 <= temp or i4;
END data_flow;
```

NPTEL (C) P. R. Panda, IIT Delhi, 2017 27

The rules are still the same with respect to triggering and execution even if I have signals here as opposed to ports, I had ports here earlier right. So, these two signal assignments where two output ports, but now I have a signal assignment to a signal. So, actually they are all called signal assignments the way to understand this is that they are all signals ports, are also thought of as special kind of signals. But in general they are all signals and the rule that we are talking about for triggering an execution of the concurrent assignment statements applied to signals in general and therefore, to also to ports in particular right.

So, this one would now be executed when i 1 or i 2 or i 3 any of them change now we have restructured the specification though which means that now first of all there are three statements instead of two, but also these two statements would now have a different rule for triggering or for execution right. Because the earlier I said that O 1 which was a function of i 1, i 2, i 3 would be triggered if any of those changes. Now we are saying that one would be triggered only if i 2 changes, but also if temp changes. So, somehow there is some difference now in the sequence of the triggering and the execution. So,

there is a difference, but is it still equivalent you think, equivalent with respect to the functionality with respect to the output.

Student: Yes, sir.

It should be equivalent yeah. So, it is just that i_1 no longer triggers O_1 and O_2 directly which it did now it triggers only temp, but that is all right because the changes any changes in i_1 are captured first in temp and if temp did not change as a result of i_1 changing did something go wrong. These expressions for O_1 and O_2 would not be triggered because temp did not change that is the rule which is ok, right because that expression sub expression did not change and therefore, the whole expression also will not change at so.

Student: Temp has a variable relation such that two of them are changing, but the output is not changing i_1 and i_2 and i_3 , suppose if i_2 and i_3 both of them change.

The rule is if any of them changes then that statement is executed, which means if two of them changed it would also be executed. All three change then it would be executed.

Student: (Refer Time: 36:05) then temp will be executing.

Right, that statement would be executing. It may result in a change in value of temp or it might not result in a change in value.

Student: Finally, the temp value may not be change in some place.

Right.

Student: But in the next statement if the complier assumes that temp has not changed.

Since, the value of temp did not change these other two statements would not even be triggered these other two assignments would not even be triggered. But it is we are with that right it is all right. This output does not change, it change in input was such that it did not ultimately propagate to the output, that is all right. Functionality wise as you can see we did nothing we just restructured the specification in a way that we avoided some things we simplified the design in some ways that do you see that there was some common part that was taken out so it is in some ways easier to specify.

If one were to directly translate that specification into a net list then this would lead to a smaller net list because there are some part of the functionality that did not repeat. What is the netlist? What we have here is an example of a netlist. Instantiations of the gates and their interconnections that is what is called a netlist.

Student: Sir, what is the circuit?

Same thing yeah, circuit is the same thing this is just the terminology that was defined in the first class, but I think you might not be there yeah.

Student: Sir, this case now temp and output one are not concurrenting all the cases.

So, they are concurrent. These three statements are specified in a way they are interpreted in a way that they are all concurrent with respect to each other. Now, but what it means is that they are not always executing each is executing when its condition is satisfied for execution that is the meaning of concurrency here yeah.

Student: (Refer Time: 38:09) so, but would there be a timing difference between the two implementation that is showing here one.


Yeah.

So, yeah the timing is an interesting topic, but we will get to that soon. As of now this is only a functionality distinction and we are saying there is no distinction they are equivalent with respect to functional. We do need to resolve we have only informally pointed out what the rule is, but it is not obvious how exactly it is going to be applied. So, but that is the topic, so why do not we just move to the next slide. A modeling of delays is what this question was about.

(Refer Slide Time: 38:43)

Contents

- Introduction
- Signal assignment
- Modelling delays
- Describing behaviour
- Structure, test benches, libraries, parameterisation
- Standards



(C) P. R. Panda, IIT Delhi, 2017

28

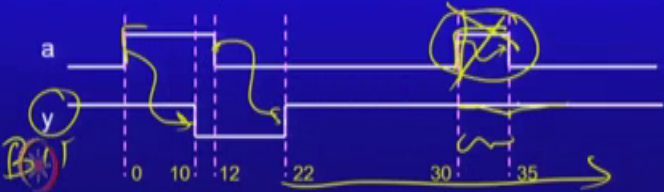
We just need to make sure that our rule is such that if multiple statements are executed and there is a dependency between those statements say which it was in that other case then we should still faithfully capture what the intention was.

(Refer Slide Time: 38:58)

Modelling Delays: inertial delay

- Models gate delays
- Spikes suppressed

```
y <= INERTIAL NOT a AFTER 10 ns;  
y <= NOT a AFTER 10 ns; -- inertial delay is default
```



(C) P. R. Panda, IIT Delhi, 2017

29

Let us talk about the delay specification. There are a couple of interesting models of delays that are supported in his HDLs and both of them let us first informally understand what is the requirement, what is it that we would like as designers and then also look at it from the point of view of a simulator that reached the HDL and reads the statements like

that and make sure that the simulator is able to capture the intent, give us the kind of functionality that we want.

The first one is what is called an inertial delay. The idea is to capture transistor related gate related delays. There is a propagation delay that is involved when the input changes in a gate and the output is triggered by the input that does not happen in 0 time. The statements that we had so far the outputs were responding in 0 time, I did not say anything about time there, but we do not need to specify time and so that means, the element of the delay comes into the picture. So, I say y takes the value of NOT a this is just an inverter specification with a couple of other things I say it is an inertial delay we will define what that is and we say that the result shows up on the output port after a certain amount of delay. So, that 10 nanosecond is the amount of delay that is what we have specified nanosecond and after an inertial these are syntactical constructs of the language.

Actually the default delay if you do not specify anything is actually inertial. So, even if you do not specify inertial you get a statement like that that is equivalent to inertial delay. Inertial as opposed to what we will get to later, but that is the requirement. What are we trying to model here? If I say that y is equal to NOT a AFTER 10 nanosecond I am trying to model an inverter whose delay is 10 nanoseconds. So, that is what I try to model which means that this kind of a waveform is what is the implication here. Let us say the waveform for a was like that that is the input right input port y is the output port; a changes at 0 at time 0. So, what we are showing here is the increasing time and y changes as a result of a changing 10 nanoseconds later. Then when a change the value of a changed again at 12 then another 10 nanoseconds later I expect the y to get the NOT a value. So, that is the requirement. But what happens if at a particular time at time 30 I have a changing, but a changes back within 5 nanoseconds that is shorter than the specified dealing, what should actually happen?

Student: (Refer Time: 42:13).

We are trying to model an inverter with a gate delay of 10 now it happened that this is more like a spike that is that is there on the input the input is changing too fast. What would the real waveform look like on the output when this happens? Of course, you see that this is a digital waveform this is; obviously, approximating what the real waveform

would be, but in reality you would have that because the input change from 1 to, from 0 to 1 the output would start falling right and at time 5 what has happened.

Student: (Refer Time: 42:50).

Have not fallen all the way down to the level two logic level that we would recognize as logic 0. So, we are somewhere here in the middle, but a goes down to 0. So, what should actually happen on the output?

Student: It should again start.

It should start rising back to this. This is what would happen in a more detailed simulation if you were to do a transistor simulation. But at this level of abstraction the way we are capturing this is that that pulse that spike would be ignored. So, nothing happens on the output. That is how the inertial delay works. It is an approximation to what the actual waveform would be, but informally the way we understand this is that that there is a delay specification for a function in our model and if the output is changing too soon right before that much time has elapsed then nothing shows on the output right. So, it is not as though it begins to fall and then gets back because then it is not it purely digital simulation, is not a 0 1 simulation remember; what is the type for y, we declared the type in our design what.

Student: Bit type.

It was a bit type right. That bit type is forced by the language to carry only 0 and 1 values right. So, the opportunity for me to show a wave falling and rising in the intermediate levels between 0 and 1 do not exist that will for that type it does not exist. So, the initial delay specification is that we will completely ignore the change due to the rising. So, this part of it whatever happened on the input we will ignore and we will just recalculate what happens as a result of the new value. So, on the output nothing will show that is an important specification of a delay type. You might not want this kind of a modeling for all your designs, but a lot of the designs but particularly there is a gate delay if you are trying to model this is considered an a reasonable approximation in a bit level simulation.

There is the second kind of delay called transport delay, that models a slightly different situation both of these are there the designer has to choose the right kind of specification

delay modeling, but this is what would be used inertial delay would be used if you have transistor related delays or gate related delays in your. So, for combinational circuits in fact, this is a right good way to model the latency of a system.

Student: Sir, if they need default values for this delay.

Default value is 0, if I did not specify like in the earlier statement, I did not specify a delay then you are saying that the output should take the new value after 0 time. But how it is coordinated is something that we will get to, but if you do not specify then it is a 0 delay specification. There is use of a 0 delay specification also. The use is that like if it is submitted to a synthesis tool like use, you specify the functionality you submit that functionality to the synthesis tool and you tell the tool you give me the best circuit. You do not know the delay right, the delay is known only after the circuit is built. So, you are only specifying the functionality and you did not specify you just said 0 delay, but that is treated as an input of the functionality by the synthesis tool and it gives you an output which at that time you know the dealer initially you actually do not know the delay. So, there are certainly uses of this of the 0 delay specification also.

Let us close this discussion now. We will continue with the transport delay later on.