Digital Circuits

Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 46
VHDL

So, in the next part of our course, we will be looking on a different topic which is known as VHDL which is a hardware description language. So, the way that we have learnt in this particular course, we have we have seen that if we want to design a circuit, we have to start with logic gates and draw the circuit and then to get a confidence on whether the circuit is working properly or not, we do some analysis of the circuit and try to see whether try to get the confidence that my design is correct.

But if the circuit is very large ok so or if the design is very complex, then it is very difficult to do that type of approach like if you are trying to do a complex design. So, there can be 2 ways in which you can do it, one is known as the top down approach another is known as the bottom of approach.

So, whatever we have learnt so far. So, we have learnt about the bottom of approach. So, we can start with the simplest modules that we have in the design and then we can after designing those and understanding that they are correct. So, we can go on connecting them in some fashion to connect to see that if the complex functions are realized.

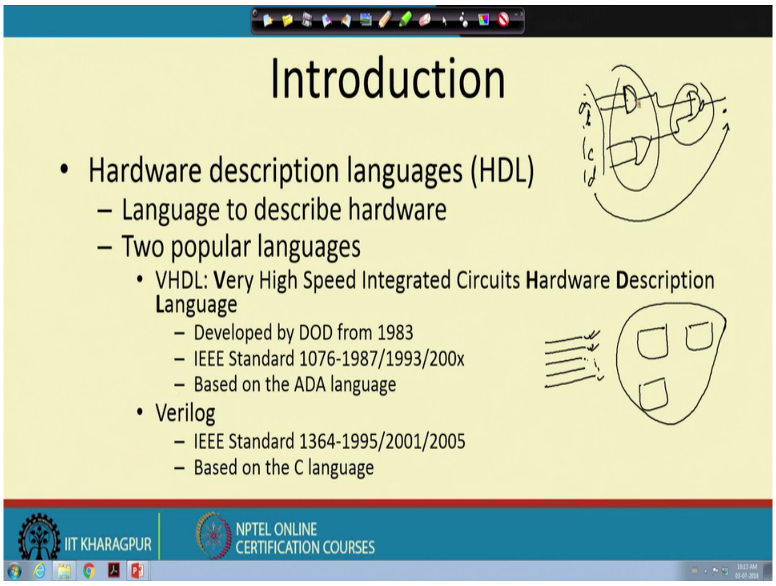
But, another approach that can be there is top down approach where we start with the top most level of the design and then go on breaking them up and coming with the further and for simpler and simpler levels of implementation.

Now, this other this top down and bottom up approaches. So, both of them; so, they becomes complex when the overall design is quite complex and in that way we need a better way to represent the represent our design or our circuit. So, this VHDL is a hardware description language that will help us in doing that and there are 2 such hardware description languages which are quite popular one is this VHDL that we are going to discuss and the another is very log.

So, both of them of equally powerful; so, there is no reason to believe that one is better than the others, it is just for an example that we have taken VHDL. So, you can very well do a do long very long as very logarithm way.

So, will be looking into a introduction of this VHDL language. So, this how does it work? What are the essential features of it? So, will we will try to go look into it by means of an example.

(Refer Slide Time: 02:45)



Introduction

- Hardware description languages (HDL)
  - Language to describe hardware
  - Two popular languages
    - VHDL: Very High Speed Integrated Circuits Hardware Description Language
      - Developed by DOD from 1983
      - IEEE Standard 1076-1987/1993/200x
      - Based on the ADA language
    - Verilog
      - IEEE Standard 1364-1995/2001/2005
      - Based on the C language

So, this hardware description languages, they come under the broad heading of HDLs language to describe hardware and so this is the um. So, they why not the standard languages that we have in software like C, C plus plus, java Pascal, Fortran, etcetera. So, those languages, why they are not suitable for this hardware the basic difference between the hardware and the software is that the hardware is always parallel in nature whereas, software is always sequential in nature.

What I mean is like this that if I write a program in say C language and these are the several the statements of the C language, then these statements are executed one after the other first this one is executed then this one. So, it goes like this until and unless there is a blanch. So, it goes in that sequence and even if there is a say blanch. So, it goes to the target and then again from the target it goes sequentially.

Whereas in a in the hardware so if I have got one functional modules, here another functional modules there another functional module here like this. Now all this functional modules so they are always active. So, whether their outputs are meaningful to the user or not that is a different question like if I have got one AND gate here and say 1 OR gate, here they are getting inputs. Now may be that the their outputs will be say mandate further to get the final output.

So, whatever be the inputs here this A B C D values. So, these gates are always workings. So, these all the 3 gates, they are always working, but we are interested about the output of this NAND gate only when this one of these values of A B C D have changed and we are looking into this part.

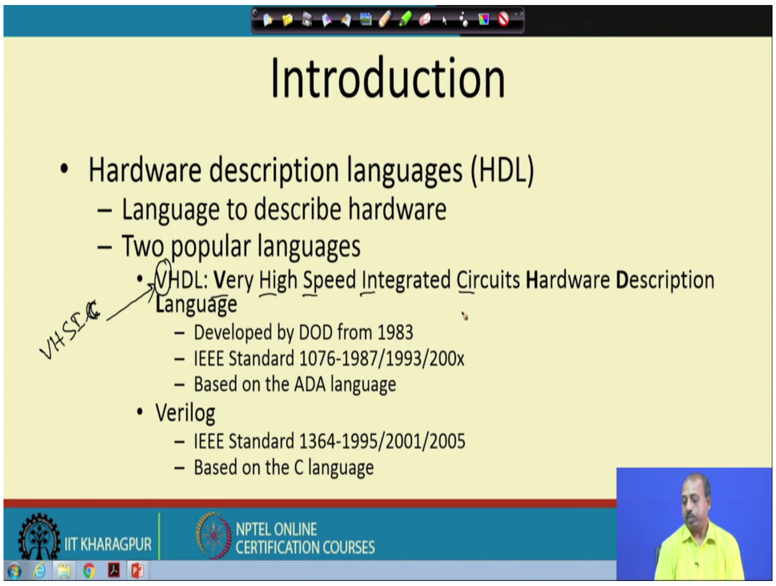
So, that way so, that is the users convenience. So, user finds it ok, I am need to know the output at this point so that is the thing otherwise if you look from this gates angle, then they are always being the competition. So, there is nothing that it is the there is nothing like sequential, it is not that this AND gate evaluates the output first, then this NAND gate it is not like that. So, all of them are working together, it is only the point of sampling at which the user is interested about the output value and that makes a difference.

So, that is why this software languages, they are not suitable for hardware because the this software languages, they will not be able to capture the parallelism that you have in hardware.

And similarly hardware languages, they are not suitable for software because hardware will assume that all the whatever we have described in that language everything is in parallel. So, it will not be able to capture the inherent sequentialism that you need in case of software.

So, coming back to the point; so, we have got this hardware description language. So, there are 2 languages; one is one is known as VHDL which is which you this V stands for this word V is stands for VHSIC ok.

(Refer Slide Time: 05:48)



Introduction

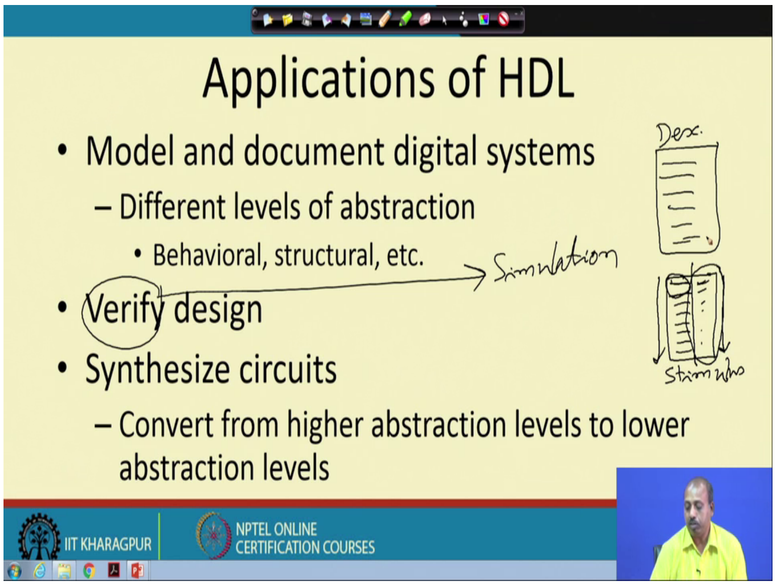
- Hardware description languages (HDL)
  - Language to describe hardware
  - Two popular languages
    - VHDL: Very High Speed Integrated Circuits Hardware Description Language
      - Developed by DOD from 1983
      - IEEE Standard 1076-1987/1993/200x
      - Based on the ADA language
    - Verilog
      - IEEE Standard 1364-1995/2001/2005
      - Based on the C language

So, VHSIC so, which is very high speed; so, this is VHSIC; very high speed integrated circuits, then hardware description language.

So, this VHSIC Hardware Description Language or VHDL; so, it was developed by department of defense from 1983, I triple E standards have come up ok. So, it is based on the ADA language. So, the basic language on which this VHDL has been designed that is ADA, on the other hand this Verilog is another standard language. So, that is again an I triple E standard has been defined and it is based on C language.

So, you can learn either of them. So, as I said that for this course will be we have taken up VHDL.

(Refer Slide Time: 06:42)


Applications of HDL

- Model and document digital systems
  - Different levels of abstraction
    - Behavioral, structural, etc.
- Verify design
- Synthesize circuits
  - Convert from higher abstraction levels to lower abstraction levels

Simulation
Desc.
Stimulus
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, why do we need this HDL? So, model and document digital systems, actually, the point at which it started is like this. So, there are large amount of designs that were there large number of circuits that were available, but the original designers are missing. So, original designers are not there.

Now, how to understand? So, like if I if I draw a complex circuit and I am not available to explain it. So, it is very difficult to understand the how what is the motivation how is this thing designed and how to preserve this design, how to um how to keep a note of the design content.

So, that is the modeling and documentation of the digital system. So, it originally started with the documentation of the digital system, we have got a circuit. So, you want to make a document to describe the circuit. So, naturally you need to the say that these are the gates in my circuit I am connecting it like this. So, this is this has to be done in some formal way.

So, for documenting this digital system; so, we have to have this HDL was necessary, sometimes, we need to model the behavior of a circuit. So, for that also this HDL is necessary and this mod there are different level of abstraction. So, we have got this behavioral level, structural level, etcetera. So, you can describe a circuit in terms of its behavior for example, if there is an ADAR so you can simply say A equal to B plus C that describe the behavior of the ADAR or you can say that my circuit contains so many

AND gate, OR gate, XOR gate, etcetera and they are connected in this fashion. So, that also defines and ADAR only, but in a different way that is the structural way of defining it.

So, the same circuit we can have a behavioral description, we can have a structural description, second objective is to verify the design. So, once you have got a description of the design so you can you can try to verify the design by means of some sort of simulation or may be by means of some formal verification technique you can try to extract the meaning of the description.

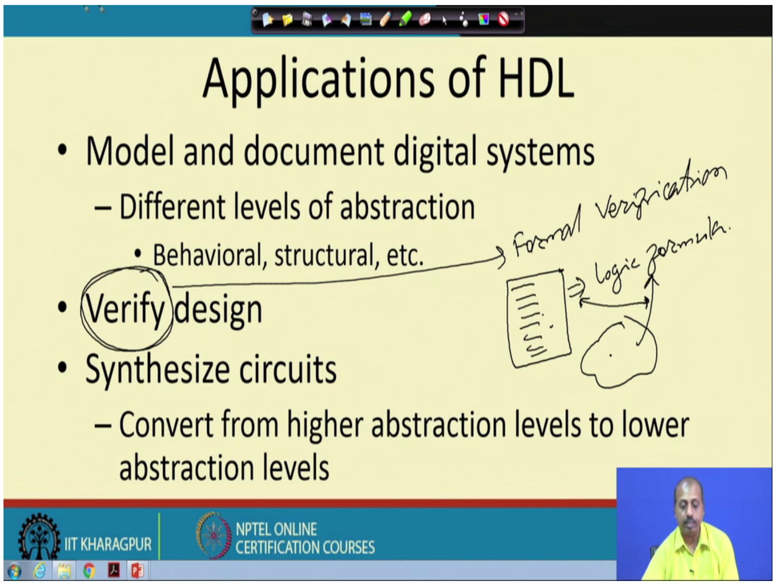
So, when you do that so you can get the meaning of it. So, that was the another that was another objective. So, this documentation was the first objective, modeling was the second objective, the third one is the verification of the design and of course, verification may be of 2 types. So, when I say I want to verify, then there can be two approaches one approach is via simulation one approach is via simulation. So, simulation means we have got a description of a system, once I have described my circuit in terms of some language and we also so this is the description and we have got some expected behavior of the circuit and for that we design something called a stimulus.

So, this is these are define some certain stimulus and we have got some expected behavior for the for each of this stimulus, we have got some expected behavior. Now what you can do you can try to apply the first stimulus onto the circuit and see whether it gives me the desired response and then you apply the second stimulus and see whether it gives the desired response only. For a combinational design so, you can apply this stimulus in any order. For a sequential design so, you have to apply then in the in a proper order only. So, whatever it is. So, if I if I apply all these stimulus then I am expecting that the response from the circuit will be like this only.

So, if that thing happens, then I get the confidence that possibly my design is ok. Now I say possibly because it is not it is not possible to enumerate all different situation that that the circuit inputs can take because that way the number of inputs that will be needed to be fed will be very high, for example, if there is a 10 input circuit, then if you are to and a combinational circuit only then if you are trying to apply all possible pattern. So, it will be 2 power 10,024. So, that is manageable, but as the number of inputs becomes higher than say 30 or 40. So, it becomes practically infeasible to apply all those patterns to the circuit and see the output.

And that the situation is more complex when you have got sequential circuits because then you have got even larger number of states that will come into picture. So, one type of verification is by the simulation. So, you have apply some stimulus to the description and try to see whether that stimulus is sufficient for, response is that you are getting from the circuit is as per your desired one. The other way of verification is by means of some formal method.

(Refer Slide Time: 11:41)



Applications of HDL

- Model and document digital systems
  - Different levels of abstraction
    - Behavioral, structural, etc.
- Verify design
- Synthesize circuits
  - Convert from higher abstraction levels to lower abstraction levels

So, this verification can be done by means of some formal methods which come under the broad heading of formal verification. So, there you try to describe the behavior. So, if this is the description of the behavior you try to describe this behavior in terms of some logic formula, you try to describe the behavior in some logic formula.

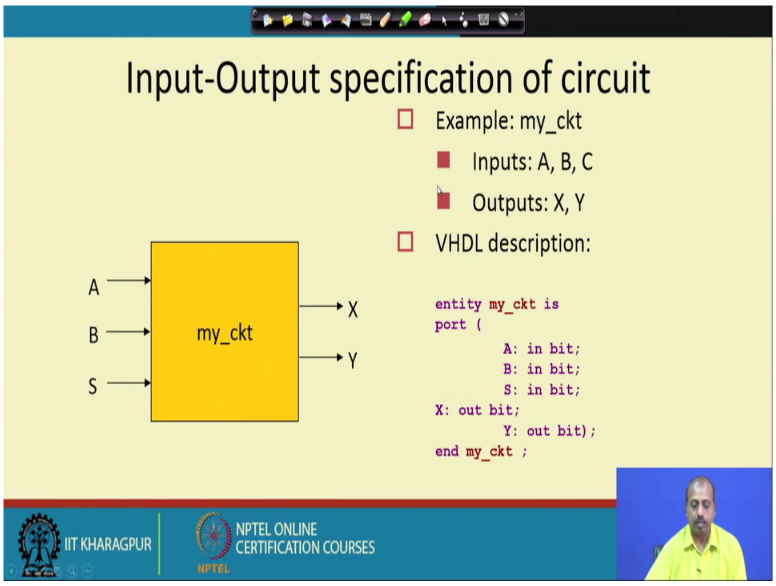
And you also have the circuit in your hand. So, if this is the circuit that you have. So, for the circuit also you can try to extract the logic formula and then try to establish an equivalency between these two logic formula ok, if we if this logic formula, they are equivalent to each other; that means, this circuit realizes this description properly, if there is a gap, then it is not shown.

So, we can do this verification by means of this formal method also, but whatever we do so we need a description of the system first. So, that is the another application area of this hardware description languages.

The third one that has become more popular now, but it was initially not that way is to synthesize the circuit. So, we start with a high level description of the circuit and then from their we convert it into lower level of obstruction. So, high level of obstruction may be a some language description of the circuit, that language description may be converted to say module level description like consisting of say adder, subtract, registers multipliers ok. So, like that. So, that can be converted into a next lower level and that from that adder subtract level description. So, we can again convert into gate level description consisting of gates and flip flops. So, that gives the next level of abstraction.

From there it can be further defined to transistor level description so that way, we can go to lower and lower abstraction levels and as we go there. So, we can and this process can be made automatd. So, if we can automate that process; that means, we have got a methodology to design a circuit from its specification. So, that, so initially it was not the major goal of these languages, but now this has become one of the major goals. So, the simulation and this synthesis; so, these are the two major goals that we have from this HDL that we have for this HDL is being used.

(Refer Slide Time: 14:15)



Input-Output specification of circuit

- Example: my_ckt
  - Inputs: A, B, C
  - Outputs: X, Y
- VHDL description:

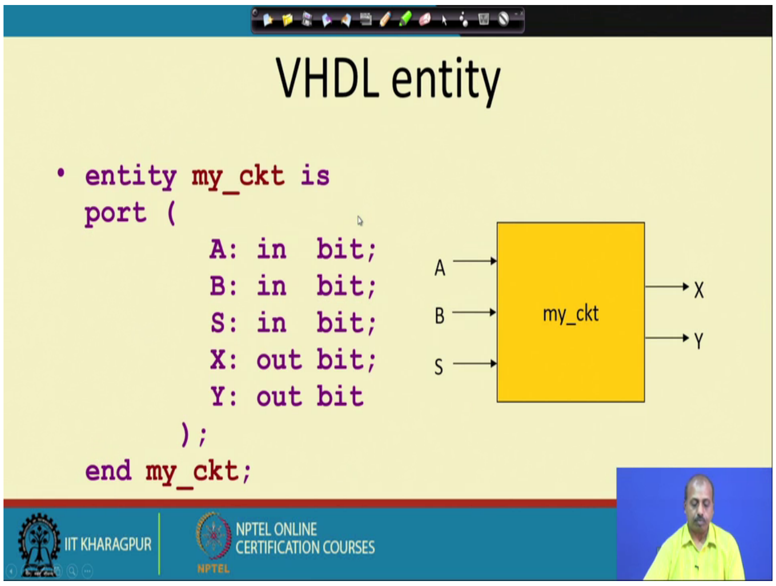
```
entity my_ckt is
port (
         A: in bit;
         B: in bit;
         S: in bit;
X: out bit;
         Y: out bit);
end my_ckt ;
```

So, next will see like; how can I specify a circuit. So, suppose this is a circuit. So, name of the circuit is my circuit. So, it has got 3 inputs, A, B and S and 2 outputs X and Y. So, this example my circuit it has got inputs A, B and C and it has got outputs X and Y. So, when you are giving it a VHDL description. So, we write it like this so this entire circuit so this will be called an entity.

So, in VHDL every module that you have will be called an entity and entity will have some ports by which it can interact with the outside world and accordingly, we can it will be the behavior of this entity has to be describe in either structurally or in a behavioral description, but the this is the interface part. So, when you are you are defining the entity from the outside how does it look like. So, one important part here is the port. So, it has got a few ports this A, B, S, X and Y. So, these are the ports and out of that A, B and S; they are input port and they are of type bit and this X and Y, they are output port and they are also of type B.

So, you see a number of key words in this description. So, for entity is a keyword port is a keyword, then this in out. So, these are keywords and this bit they are actually the data type or the signal type. So, A, B, C can say they are called A, B, S, X. So, as they are signals. So, they are either input output or it can be bidirectional also. So, you can have this type as in out or mode as in out and their type is bit. So, all of them are of type bit. So, you have got this. So, in this way you can define entities.

(Refer Slide Time: 16:09)

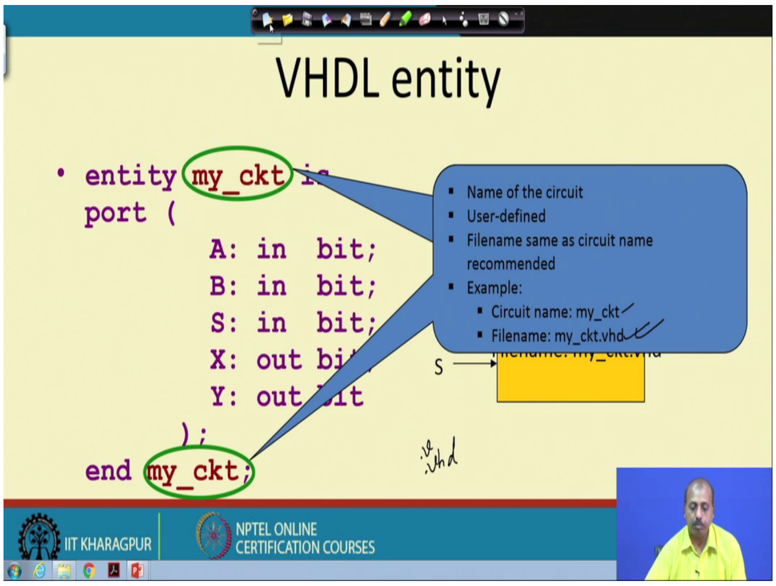


VHDL entity

```
• entity my_ckt is
  port (
          A: in  bit;
          B: in  bit;
          S: in  bit;
          X: out bit;
          Y: out bit
        );
  end my_ckt;
```

So, this is the VHDL entity; my circuit is port A, B, S and these are input port and type B, X and Y, they are output port of type bit ok.

(Refer Slide Time: 16:24)



VHDL entity

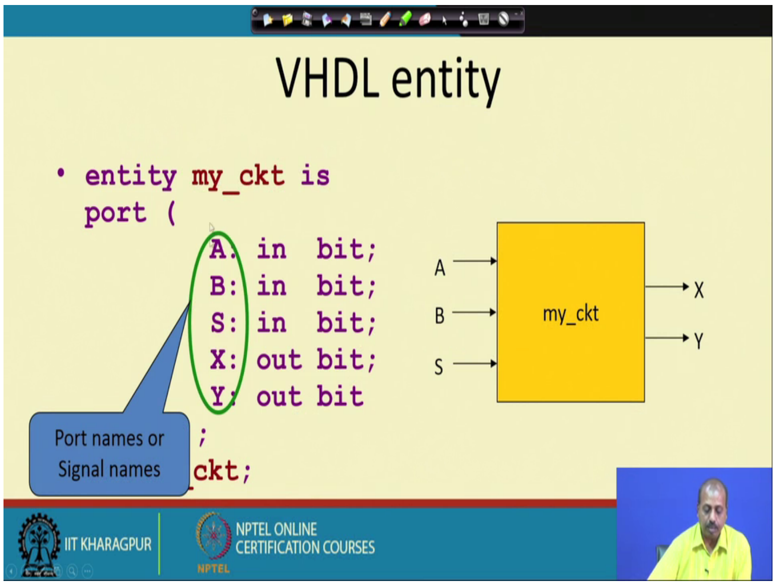
```
• entity my_ckt is
  port (
            A: in  bit;
            B: in  bit;
            S: in  bit;
            X: out bit;
            Y: out bit
         );
  end my_ckt;
```

- Name of the circuit
- User-defined
- Filename same as circuit name recommended
- Example:
  - Circuit name: my_ckt
  - Filename: my_ckt.vhd

So, the things to be noted the my this my circuit, this is the name of the circuit. So, every circuit should have entity should have a name. So, in a one description, I can have only 1 name for entity so and the names are unique. So, if there are multiple entities so they should have different names, but these names are unique. So, this is user defined. So, this names user can give. So, file name same as circuit name. So, this is A; this only recommended. So, this description in whatever file name your file, you are writing. So, normally we write them in files having extension dot v or dot vhd like that.

Many tools; they will acts a different types of extensions. So, for this case it may be file name may be my circuit dot vhd circuit name is my circuit and file name is my circuit dot vhd, but this is just a just a suggestion or recommendation, but you may or may not follow it ok. So, that may or may or may not follow. Now next will be this.

(Refer Slide Time: 17:33)



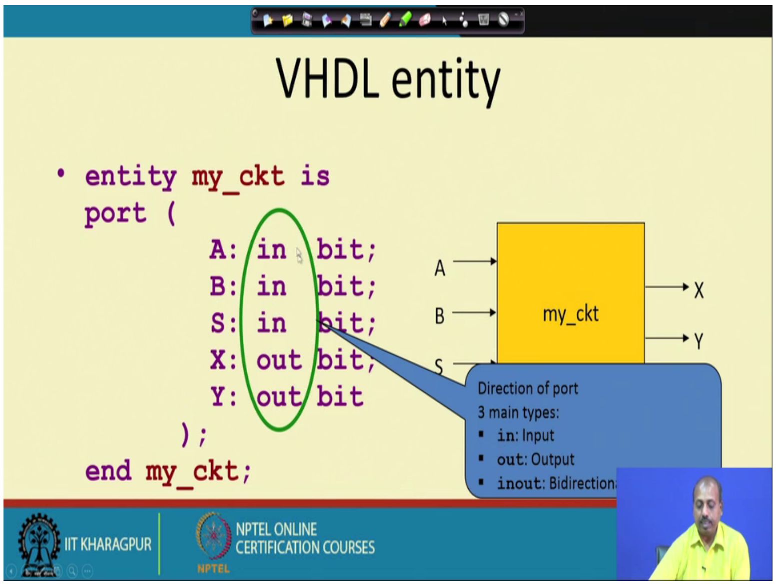
VHDL entity

```
• entity my_ckt is
  port (
          A: in  bit;
          B: in  bit;
          S: in  bit;
          X: out bit;
          Y: out bit
        );
  end my_ckt;
```

Port names or Signal names
A
B
S
my_ckt
X
Y
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

These are the port names A, B, S, X Y. So, they are port names or signal names. So, they are the interfacing signals that we have again the names here can be arbitrary. So, they can be chosen by the user, but again the same thing that the same name cannot repeat ok. So, that restriction will be there.

(Refer Slide Time: 17:54)



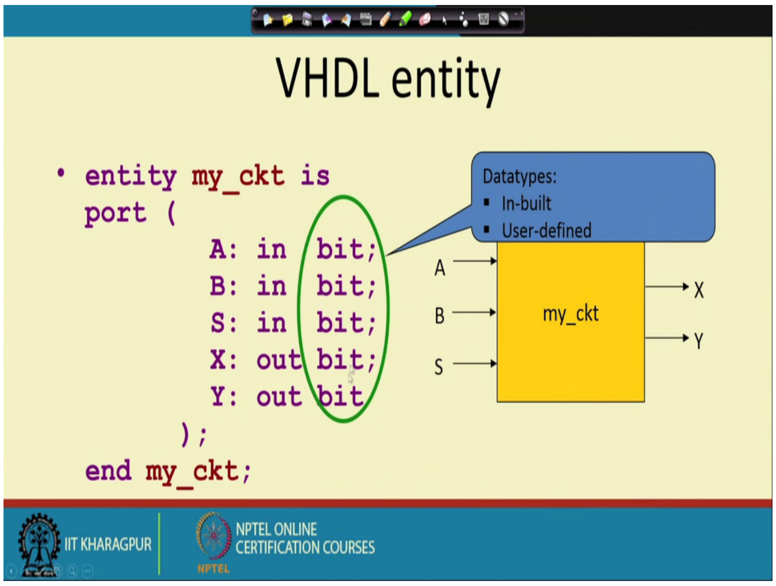
VHDL entity

```
• entity my_ckt is
  port (
          A: in  bit;
          B: in  bit;
          S: in  bit;
          X: out bit;
          Y: out bit
        );
  end my_ckt;
```

Direction of port
3 main types:
- in: Input
- out: Output
- inout: Bidirection

Then the next part is the direction part. So, this in and out; so, they are directions. So, there can be three main direction types one is in for the input out for the output and in out for the bidirectional; so, in out and in out. So, these are the bits ok.

(Refer Slide Time: 18:16)



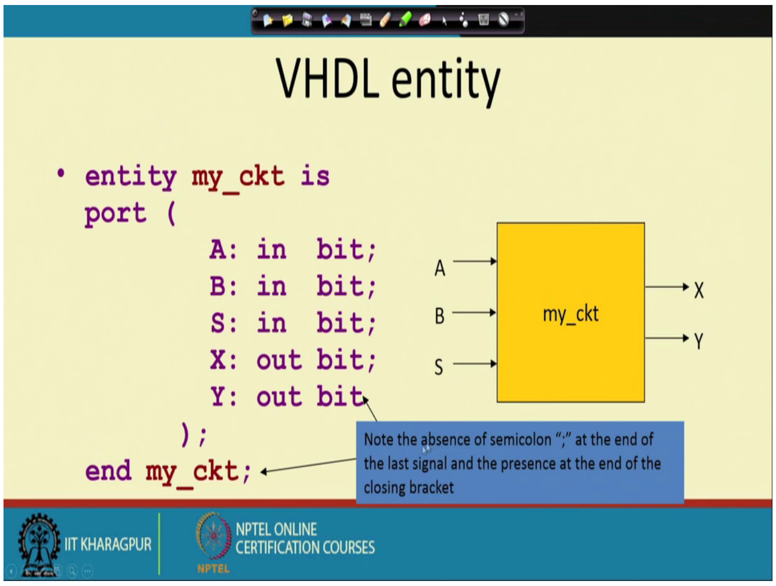
VHDL entity

```
• entity my_ckt is
  port (
          A: in  bit;
          B: in  bit;
          S: in  bit;
          X: out bit;
          Y: out bit
      );
  end my_ckt;
```

Datatypes:
- In-built
- User-defined

Then comes the data type. So, these are the data type. So, they all of them are bit. So, there are some built in data type just like any programming language you have got data types like say integer character Boolean real like that. So, here also this HDL language, it has got built in data types like bit is A built in data type, the other built in data types, they are like integer byte etcetera they are they are, but they can be that I can have some user defined data type also that is also possible.

(Refer Slide Time: 18:50)



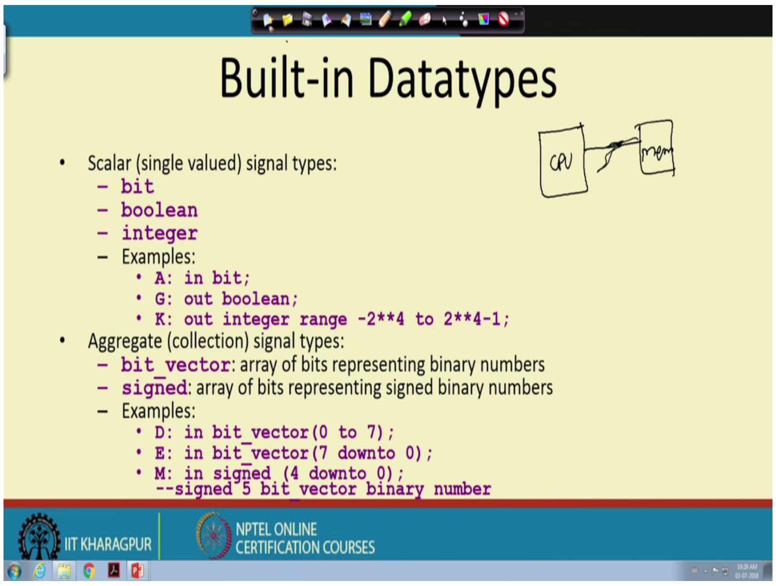
VHDL entity

```
• entity my_ckt is
  port (
          A: in  bit;
          B: in  bit;
          S: in  bit;
          X: out bit;
          Y: out bit
      );
  end my_ckt;
```

Note the absence of semicolon ";" at the end of the last signal and the presence at the end of the closing bracket

So, none; so, the for the as far the syntax is concerned there is a small observation that the on the last port we do not have the semicolon last signal, we do not have the semicolon and the semicolon is put at the end of the circuit description the presence at the end of the closing bracket. So, this one so you see that this the there has to be a semicolon here. So, both the as if this semicolon is taken from here and put at the end of this closing bracket; so, this way we can define a VHDL entity. Next we will be looking into the built in data types.

(Refer Slide Time: 19:30)



Built-in Datatypes

- Scalar (single valued) signal types:
  - bit
  - boolean
  - integer
  - Examples:
    - `A: in bit;`
    - `G: out boolean;`
    - `K: out integer range -2**4 to 2**4-1;`
- Aggregate (collection) signal types:
  - bit_vector: array of bits representing binary numbers
  - signed: array of bits representing signed binary numbers
  - Examples:
    - `D: in bit_vector(0 to 7);`
    - `E: in bit_vector(7 downto 0);`
    - `M: in signed (4 downto 0);`
      `--signed 5 bit_vector binary number`

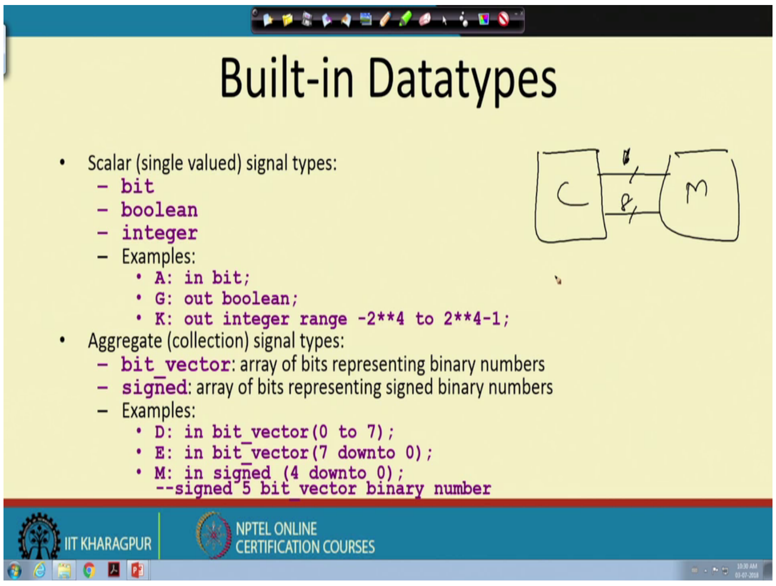
So, first the scalar or single valued single valued signal types bit boolean integer ok. So, these are the different single valued signal types bit, they can be getting the value 0 or 1 boolean can be true or false, integer can be there will be some range ok. So, you can also tell the range of the integer. So, normally you can there can be this unbounded integer.

So, in that case the system will a assume some bound otherwise you can also tell the range of the integer so which is not possible in the software programming language using most of the cases. So, we do not have this range available, but here we have got the range available. So, here we can say that a is an input port of type bit G as a output of type Boolean and K is an output of type integer with the range minus 2 to the power 4 to 2 to the power four minus 1. So, it is basically 4-bit signed representation like that ok.

So, these are the single valued things for the aggregate one or the collection. So, we have bit vector because most of the times in digital design so we will be having connections.

So, which are arising nature for example, if we are if we are having this say CPU and memory; CPU and memory, then we have got this at this bus data bus type of connections.

(Refer Slide Time: 21:16)



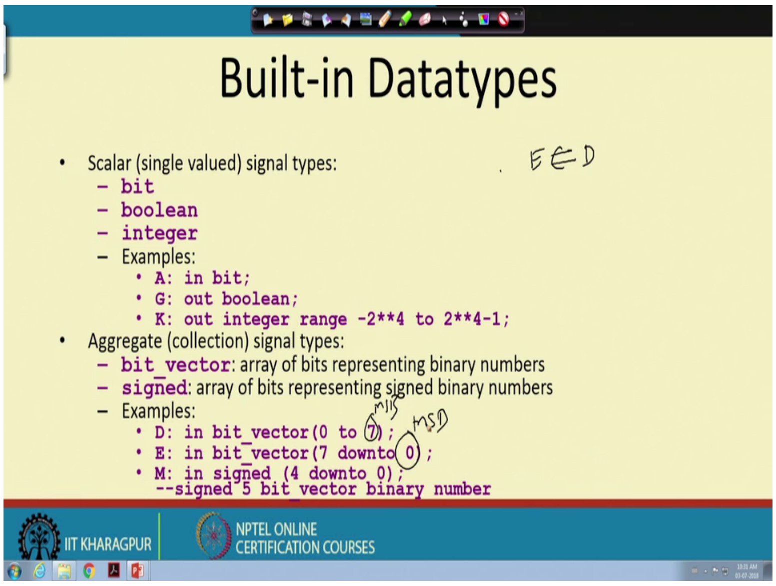
Built-in Datatypes

- Scalar (single valued) signal types:
  - `bit`
  - `boolean`
  - `integer`
  - Examples:
    - `A: in bit;`
    - `G: out boolean;`
    - `K: out integer range -2**4 to 2**4-1;`
- Aggregate (collection) signal types:
  - `bit_vector`: array of bits representing binary numbers
  - `signed`: array of bits representing signed binary numbers
  - Examples:
    - `D: in bit_vector(0 to 7);`
    - `E: in bit_vector(7 downto 0);`
    - `M: in signed (4 downto 0);`
      `--signed 5 bit_vector binary number`

So, we say this CPU and memory. So, we have got this at this bus and data bus type of connection. So, this these are may be 16 bit ok, some other data bus may be eight bit like that. So, this way there are a number of bits which are constituting each signal line. So, the they act in instead of treating them as individual bits. So, it is better to treat them as a vector. So, that is the bit vector.

So, we have got this bit vector as a we have got this bit vector as a collection. So, array of bits representing binary numbers. So, that is bit vector it can be signed. So, can array of bits representing signed binary numbers like say we have got D in bit vector 0 to 7. So, this is D is an input port where there are eight lines and the lines are named as 0 to 7. So, D 0 to D 7, we have got 8 bits here, similarly, E is input, it is again of type bit vector and 7 down to 0. So, if this is so we have got E 7 2 0. So, if you are assigning say if you also; so, the one is 2 another is down 2. So, if you are writing like say E assuming that E is out port output port.

(Refer Slide Time: 22:39)



Built-in Datatypes

- Scalar (single valued) signal types:
  - bit
  - boolean
  - integer
  - Examples:
    - A: in bit;
    - G: out boolean;
    - K: out integer range -2**4 to 2**4-1;
- Aggregate (collection) signal types:
  - bit_vector: array of bits representing binary numbers
  - signed: array of bits representing signed binary numbers
  - Examples:
    - D: in bit_vector(0 to 7);
    - E: in bit_vector(7 downto 0);
    - M: in signed (4 downto 0);
      --signed 5 bit_vector binary number

E ⇐ D

MSB

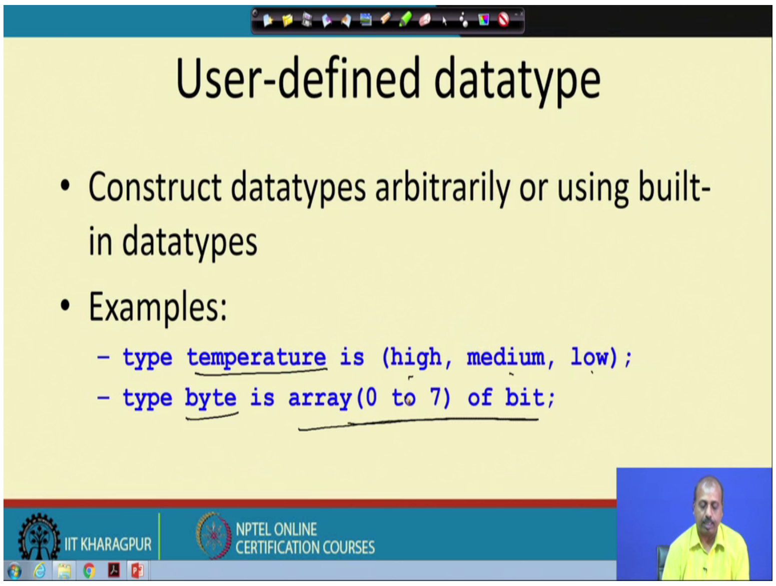
MSB

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, if you are right like say E should get the value D, then this D 0 will come to E 7 like that because this is the MSB here is the. So, this is the MSB here, but in case of E down to 0. So, this is going to be the MSB.

So, that way there, there values will be assigned properly then this 4 M in sign 4 down to 0. So, a sign 5 bit, 5 bit vector that is binary number ok. So, we can have different types of representations.

(Refer Slide Time: 23:19)



User-defined datatype

- Construct datatypes arbitrarily or using built-in datatypes
- Examples:
  - type temperature is (high, medium, low);
  - type byte is array(0 to 7) of bit;

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Also we can have the user defined data type. So, this construct we can construct data types arbitrarily or using built in data or by using built in data types for example, we can say type temperature is high medium low or type byte is array 0 to 7 of bit. So, this so we are defining two different types here the first one is defining the type temperature. So, and its possible values are high medium and low. So, the second one is defining a type called bite and it says that this is an array 0 of 7 of bit ok.

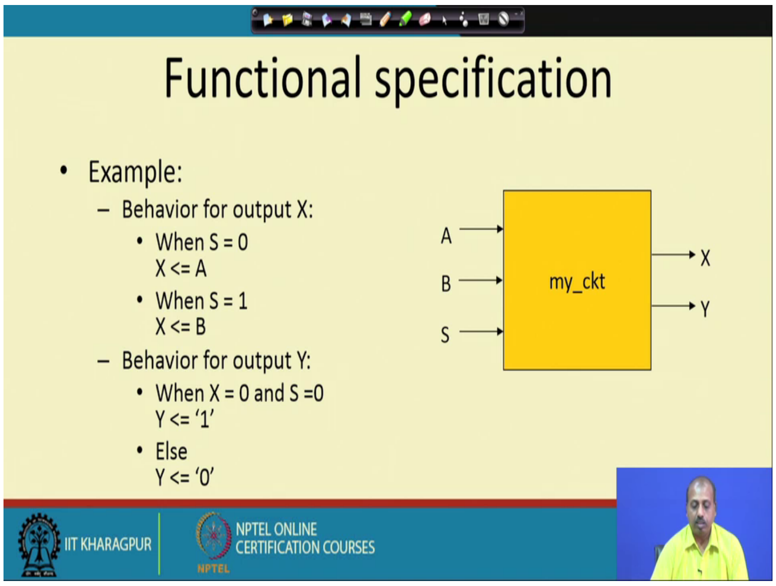
So, next if you say that I have got a signal X of type byte. So, this will mean that this is an array of bits 0 to 7 or if you have got the say T of type temperature. So, it can take up the value high medium and low. So, apparently it seems that this the user defined type. So, why are you introducing this say other types like set temperature as a variable temperature as a type where these values are like this high medium low like that.

So in fact, if you look into this VHDL manual you will find that it will allow you to define any user define types, for even you can define the physical quantities like the size of a size of a system, then its weight and all. So, which are not at all related to its electrical quantities like say temperature of a body. So, that is not related to the electrical signals within that for that system.

So, but still they can be described. So, while describing a physical system. So, you may take help of those those special types special data types and use them for your description so that actually makes that these are the spaces the description complete for the completeness of the description. So, you can use those things though they are not synthesizable for example, if you have got a type temperature high low medium then there is no way by which I can do a synthesis of this temperature in my circuit. So, that is not possible.

But still this strategy has been the given the; so that we can define some physical quantities by means of this VHDL language that makes VHDL really an exceptional one compared to other programming languages where we do not have this type of features.

(Refer Slide Time: 25:51)



Functional specification

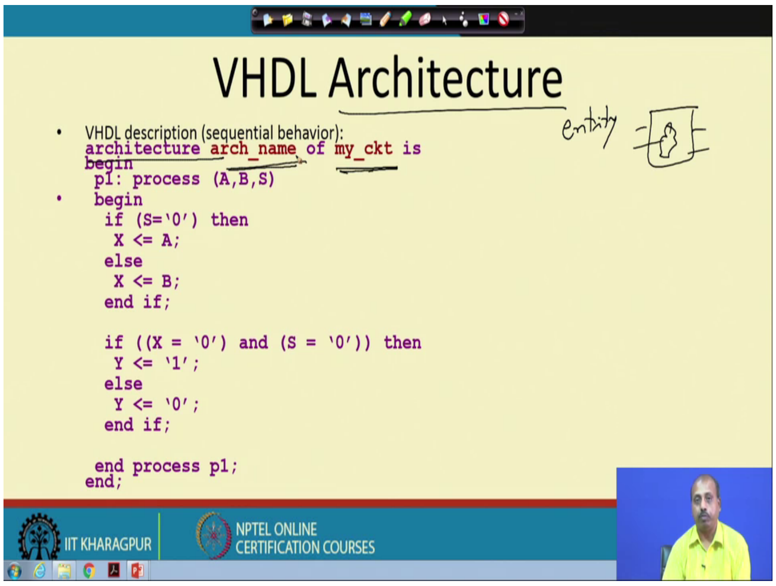
- Example:
  - Behavior for output X:
    - When S = 0
      X <= A
    - When S = 1
      X <= B
  - Behavior for output Y:
    - When X = 0 and S =0
      Y <= '1'
    - Else
      Y <= '0'

A
B
S
my_ckt
X
Y
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Next will be looking into the functional specification; so, functional speciation like we can say that behavior for output X. So, we can say that when S equal to 0 X is assigned A and when S equal to 1 X is assigned B. So, suppose this is the behavior of X and behavior of Y is when X equal to 0 and S equal to 0, then Y is equal to 1 else Y equal to 0.

So, this may be the description of my circuit. So, behavior of X and Y from the from the English language description of this my circuit may be we know that this will be the situation then the when S is 0, then X will get A when S is 1. So, there is a multiplexor here. So, between A and B; so, when and the select line is S and then it is X and if X is 0 and S equal to 0, then y is 1 otherwise Y is 0. So, that way that is the description.

(Refer Slide Time: 26:52)



VHDL Architecture

- VHDL description (sequential behavior):

```
architecture arch_name of my_ckt is
begin
 p1: process (A,B,S)
 begin
  if (S='0') then
   X <= A;
  else
   X <= B;
  end if;

  if ((X = '0') and (S = '0')) then
   Y <= '1';
  else
   Y <= '0';
  end if;

 end process p1;
end;
```

So, how are you going to describe it in the VHDL? So, first one will be looking into a sequential behavior. So, the any description of this VHDL is put into the VHDL architecture.

So, for my circuit the first thing that I have seen is the entity. So, entity has told over the interface with the outside world and then I am trying to describe what is there inside. So, for that I have to use this architecture part.

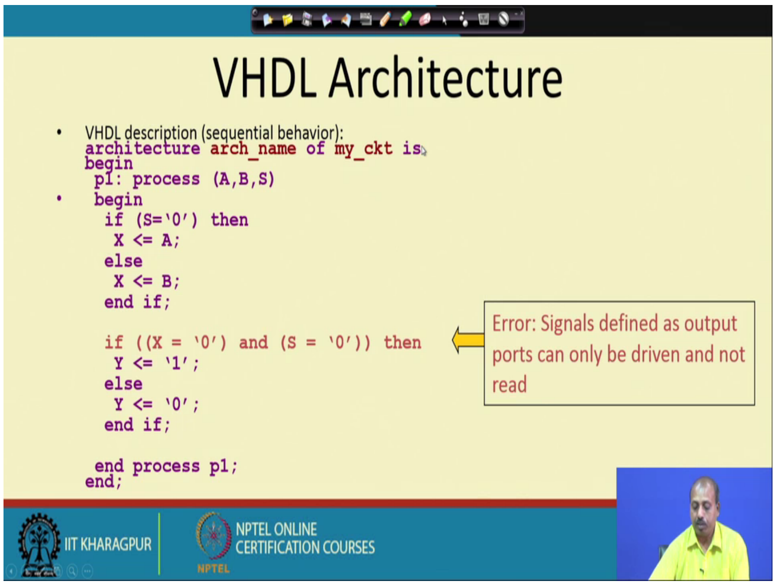
So, this architecture keyword it starts with the architecture keyword again, I can have several architectures of the same entity. So, for this entity my circuit I am defining an architecture and these are particular architecture name is say arch name. So, I can have another architecture also defined for my circuit. So, there is no restriction about how many architectures I am I am going to define for this entity because this was more from the documentation point of view while initially designed that is why it is like this.

So, you can define one architecture from the behavioral side another architecture from the structural side like that. So, that is why I can have several architectures for the same entity.

So, here we have got this architecture arch name of my circuit is and then begin. So, these are all keywords architecture of is begin, then this as oh whenever you are trying to define something sequential so you have to take help of this process block ok. So, this process, it so p 1 is a process. Now begin if S equal to 0, then X A, else X equal to B end if X equal to 0 and S equal 0, then A Y equal to 1, else Y equal to 0 end if then end process P 1.

So, this is the process description and it depends this process output depends on the signals A, B and S. So, they actually defines the values on which they are this process values are going to change. So, if there is A change in the values of A, B and S then this description has to be followed to determine what are the values of X and Y. So, that is so what is done by this VHDL architecture block.

(Refer Slide Time: 29:12)



VHDL Architecture

- VHDL description (sequential behavior):

```
architecture arch_name of my_ckt is
begin
 p1: process (A,B,S)
 begin
  if (S='0') then
   X <= A;
  else
   X <= B;
  end if;

  if ((X = '0') and (S = '0')) then
   Y <= '1';
  else
   Y <= '0';
  end if;

 end process p1;
end;
```

Error: Signals defined as output ports can only be driven and not read

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, there are some restrictions like this signals defined as output ports can only be driven and not read for example. So, this X is defined as an output port and I am trying to read the value of X here.

So, that is not possible because that is an output from the from my entity so I cannot read it. So, in this way from the it can catch many of this errors in from that description itself. So, you see that these are the these are the some of these are some of the advantages that we have with VHDL ok. So, it can catch this type of design errors very easily, otherwise, it physically means that from the output, you are again drawing a line inside the circuit to check whether X equal to 0 or not. So, that is not a very good design practice. So, that has to be avoided.