Digital Systems Design with PLDs and FPGAs

Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science-Bangalore

Lecture-42
VHDL Test bench

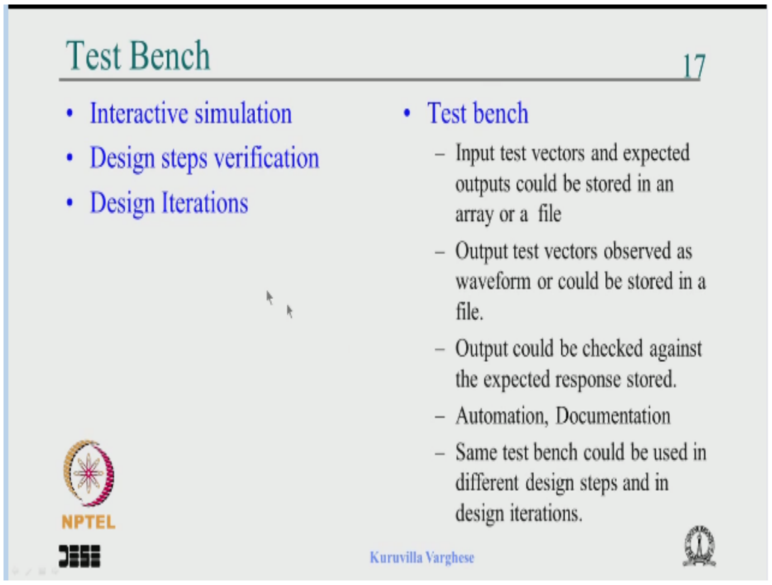
Welcome to this lecture on VHDL, test benches in the course digital system design with PLDs and FPGAs. I have covered all most the complete syllabus what is remaining is basically discussing a case study winding up everything. That means the design the VHDL, FPGA and all that. And I am planning to show towards the end in the last lecture a VHDL design tool from Xilinx.

And I will also as part of that lecture I will show the case study we have going to discuss in the next lecture on an FPGA boat. I keep this part towards end so, that is kind of de-coupled from what we have discuss because these tools change interfaces change. So, if I show as part of the lecture, then that part of the become irrelevant after a while. So, that is why I kept it towards the end.

But 1 part which is essential for writing VHDL and verifying is the test bench which we have not covered and I am not covered many things like functions and procedures and libraries in VHDL. But I am sure you will be able to back it up reading I have in covered that because is not much use in the design the functions and procedures by you know there is less need of writing functions and procedures so**.**

That I have kind of skip, but you can go through it. Similarly the libraries and packages it is a very minor kind of syntax, extra syntax and something is tools specific working you know compiling into some kind of library and all that which is not part of the VHDL the part of the VHDL is how to generate the syntax for creating package header and so on. So, today we will look at test benches which enables as to do the verification of the design we do in the tool. So, let us go into the slides.

(Refer Slide Time: 02:52)


Test Bench

17

- Interactive simulation
- Design steps verification
- Design Iterations

- Test bench
  - Input test vectors and expected outputs could be stored in an array or a file
  - Output test vectors observed as waveform or could be stored in a file.
  - Output could be checked against the expected response stored.
  - Automation, Documentation
  - Same test bench could be used in different design steps and in design iterations.

Kuruvilla Varghese

So, normally when you simulate at the beginning most of the time you do this called interactive simulation. That means you will compile your design into a library and in the simulator you apply some stimulus and see output as a waveform again you give some input then view the waveform and so on okay, it is a very kind of interactive session you keep giving the inputs, then verify the output.

And this is a very it is okay for a very simple design, but in a complex design this is quite difficult. Because the complex design will have lot of inputs, lot of outputs and to essentially verify the functionality you need a quite a lot of inputs. So, this is a long run of simulation. And that to verify from 1 end to other end maybe it will run into seconds and steps are say order of nanoseconds.

And to verify all that you know going through this very very difficult if not impossible and in addition to that you know you have functional simulation you will iterate through the functional simulation. Then you will come to timing simulation and you will iterate through that again. Sometime you go back all the way to functional simulation depending on the errors and in the optimisation you want to do.

So, in all these if you want to verify long run of manually, it is quite a difficult task. So, the so called test bench is a VHDL code which instantiate your design and apply the stimulus in kind of in a step by step manner and verify the output automated you know. That process can be automated. So, that is how test bench is useful. So, essentially what we are what I am saying is that you can store the input test factors and for those test factors what are the expected outputs in 1 place.

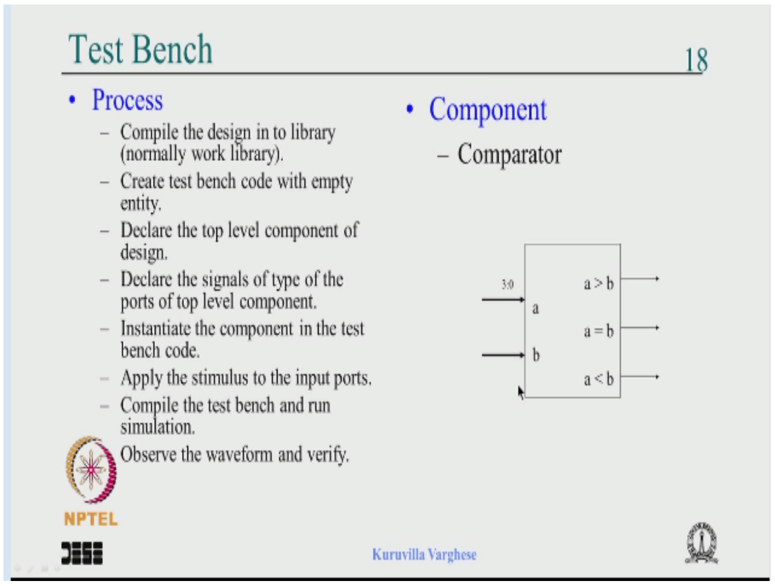
It could be in an array or a file. Then either you can view the in the into in a simulator or even better is that after applying each set of inputs you can extract the expected response stored. And you know compare against the actual output. So, step by step you apply the setup inputs then check the output is kind of equal to the expected response sometime it is called the golden result or whatever.

So, that is compared again good one you know the really expected correct one is compared normally that can be generated from a correct model of your design. So, basically it allows you to automate you know that means that you run it, it might take quite some time to run through all the test factors. But at the end if the response say that it is kind of verified.

Then you are sure that it is working you do not have to run through long run of way form and sometime you know. Suppose at some point there was a output mismatch you know what the expected output was not met by the actual output. Then at that point in the console there will be a message saying that particular output did not match the expected output.

So, you can go to that particular instance and verify, you know can check what is going wrong and you could try to you know address that particular issue. So, test bench that way automate the whole verification process. It is a must you know you cannot do without test bench in verification.

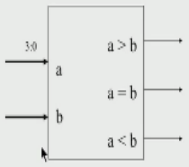
(Refer Slide Time: 07:22)



Test Bench

18

- **Process**
  - Compile the design in to library (normally work library).
  - Create test bench code with empty entity.
  - Declare the top level component of design.
  - Declare the signals of type of the ports of top level component.
  - Instantiate the component in the test bench code.
  - Apply the stimulus to the input ports.
  - Compile the test bench and run simulation.
  - Observe the waveform and verify.
- **Component**
  - Comparator



Kuruvilla Varghese

So, let us look at process in a the simple process wherein we apply the inputs you know 1 by 1. And assume that now for starting we just manually verify the output at least let us understand how we can automate the whole applying the stimulus at the input in a automated manner and we will verify this manually. So, that is assumption, so the process is that you have to compile the design into a library.

It need not be a particular you know special library it can be the work library. So, in a simulator or a compiler any tool whatever you compile gets into work library. So, you have to compile your design into work library. Now create a test bench code with empty entity. Test bench code is nothing but a VHDL code that is use for verifying the design component you have to decide okay.

Now we have no plans to implement this is not a kind of implementation code. This is just a verification code. And so that we do not need an entity with the input output port. It is not a component which we are going to implement it just we want to see input output of waveform. So, we just keep the entity empty we do not declare anything in the entity.

Now what we are going to do is that we have to instantiate this particular (()) (09:04) into the test bench code okay. then only we can apply the input you know we will instantiate that is a component probably only component you know there is no need of anything else you know you just instantiate that top level component maybe your component consist of the interconnection of various other components.

But we are not interested in that we are interested in the top level component though when you compile that into work library everything will be there in the code. But we instantiate we are going to instantiate that design into our test bench code. And so we have to declare that component, we have seen that to instantiate we have to declare the component, because it is in the work library it is not in compile into a particular library in a package and all that not required.

So, we declare the top level component of the design and now when we instantiate a component we have to connect some signals to it you know you cannot instantiate a component without connecting a signal. But we are not connecting this component to anything else. You know we just want to access the input apply the stimulus, we want to access the output just to observe the waveform.

So, it is enough if it declare the signals of the type of the ports of top level component that by that I mean if the component has say ab input and z output. We have to say there is a signal called a, a signal called b which is for which are 4 bits. And re is an signal called z which is 1 bit signal that is shall. Now the moment you declare the component, moment you declare the signals, you can instantiate that component.

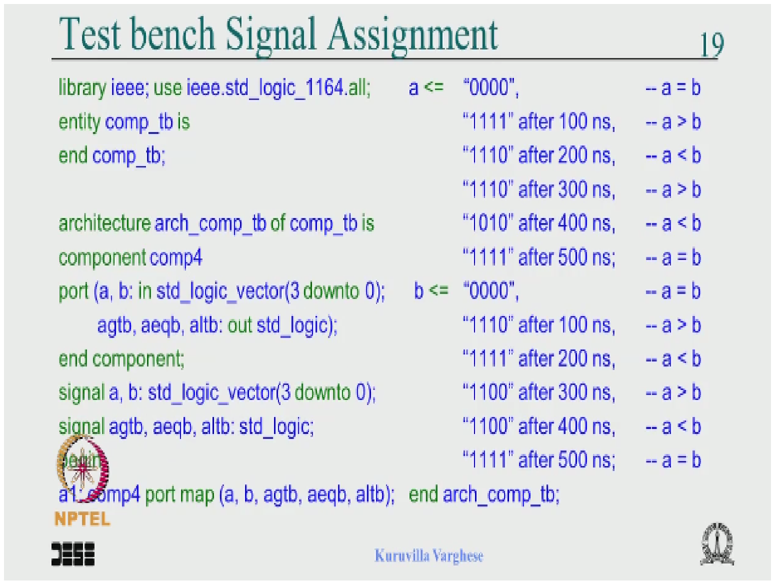
Basically in the code it is nothing but you know a is connected to signal a, b is connected to signal b, z is connected to signal z that is all when you instantiate that now to the input signal you can apply the stimulus you apply the stimulus in the code. Then the test bench code is complete. Now you can compile that test bench and run the simulation. And there will be a probably a long set off inputs which is applied.

You will get all the waveforms in one short, you can verified okay, that is the essence of the test bench code, a simple test bench codes. You compile design into library create a test bench code with entity and entity declare the component, declare the signal instantiate the component into the code apply the stimulus. Then the code is over the test bench code is over. You compile it into to the work library.

And you know and run simulation and you observe the and verify so, I am going to show that VHDL code is an example what I have written is a comparator with 2 4 bit inputs called ab which is standard logic vector 3 down to 0, both a and b. And we have 3 output, 1is a greater than b, a equal to b, and a is less than b so, all the three you know. These are kind of mutually exclusive that 1 is any time only one of them could be 1.

So, let us create a test bench code to verify the functionality or the timing of this component and initially we will concentrate on the functional simulation not the timing simulation.

(Refer Slide Time: 12:53)



Test bench Signal Assignment

19

```
library ieee; use ieee.std_logic_1164.all;     a <=  "0000",                -- a = b
entity comp_tb is                                    "1111" after 100 ns,  -- a > b
end comp_tb;                                         "1110" after 200 ns,  -- a < b
                                                     "1110" after 300 ns,  -- a > b
architecture arch_comp_tb of comp_tb is              "1010" after 400 ns,  -- a < b
component comp4                                      "1111" after 500 ns;  -- a = b
port (a, b: in std_logic_vector(3 downto 0);   b <=  "0000",                -- a = b
      agtb, aeqb, altb: out std_logic);              "1110" after 100 ns,  -- a > b
end component;                                       "1111" after 200 ns,  -- a < b
signal a, b: std_logic_vector(3 downto 0);           "1100" after 300 ns,  -- a > b
signal agtb, aeqb, altb: std_logic;                  "1100" after 400 ns,  -- a < b
begin                                                "1111" after 500 ns;  -- a = b
a1: comp4 port map (a, b, agtb, aeqb, altb);   end arch_comp_tb;
```

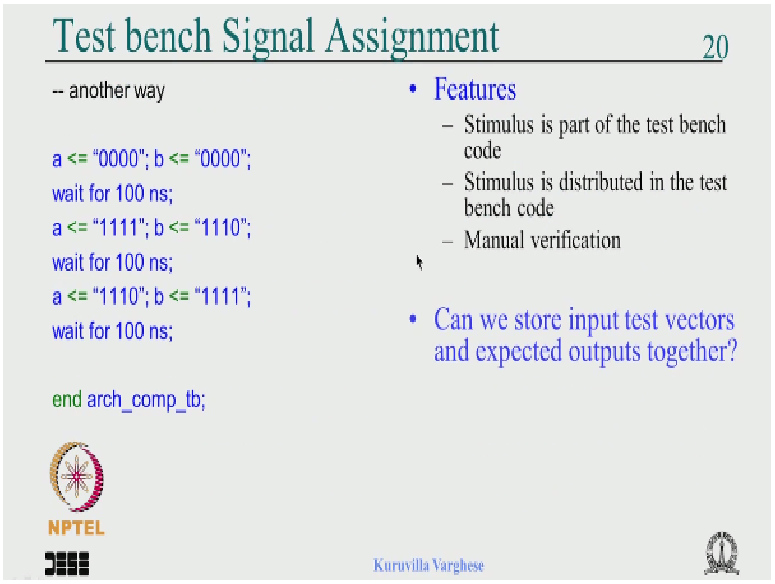
NPTEL

Kuruvilla Varghese

So, this the code you know you have the library declaration, library ieee then the package declaration use ieee standard logic 1164, all that is to be able to use standard logic data type. And you see here entity comp_tb is and you say end comp_tb okay tb have appended to show that it is test bench. You know you can use your own kind of names where whatever mode you want to use it.

You can use it whatever particular style you want to adopt, you can adopt that. So, it is an entity and entity now we write the architecture, architecture and name of this particular entity is now before begin we have to declare the component we are to declare the signal okay. So, component come for is port a b in standard logic vector 3 down to 0 okay this is the left part of the code up to here okay.

(Refer Slide Time: 13:59)



Test bench Signal Assignment

20

-- another way

```
a <= "0000"; b <= "0000";
wait for 100 ns;
a <= "1111"; b <= "1110";
wait for 100 ns;
a <= "1110"; b <= "1111";
wait for 100 ns;

end arch_comp_tb;
```

- Features
  - Stimulus is part of the test bench code
  - Stimulus is distributed in the test bench code
  - Manual verification
- Can we store input test vectors and expected outputs together?

NPTEL
Kuruvilla Varghese

And this is the right part of the code and now ab is 4 bit vector inputs a greater than b, a equal to b, a less than b is single bits out standard logic, n component. Now we when we instantiate we have to some signal to ab all this so, we declare the signals of the same name okay, it is very easy you say signal ab is standard logic vector 3 down to 0, signal a greater b, a equal to b, a less than b is standard logic exactly same.

So, the declaration is ober we say begin and now we can instantiate the component okay, we give a label and you say particular component comp4 port map and the positional association, a is connected to a, bis connected to b, greater to greater, equal to equal, less to less. So, ab greater equal less in the same order and then so, we have instantiated the component. Now to this particular wire a and b we are applying the input.

And now you see how that is done, you say a is assign all 0s that means that 0 nanosecond this is se are the comments and you say comma 1111 after 100nanosecond, 1110 after 200nanosecond, comma and so on okay. So, we have in seen this kind of syntax we have seen after some delay.

But it is possible to keep on specifying you know after 100nanosecond, after 200nanosecond and so on okay.

So, this is like a kind of sequence of inputs at the particular instant is applied to a so, we write similarly for the b only thing is that the comment say that you know both are 00 at 0nanosecond. So, it is equal and here it is 1110 so, it is a greater than b so, that is enough the VHDL code is complete. So, as I said library package empty entity component and signal declaration instantiate the component apply the waveform to the input signal okay.

Now one problem with this particular and you just compile ad run the simulation in simulator. It will apply all these and you will get the output waveform and you can verify but trouble with this code is that assume there are some 5 such of input okay. And you will have a, b, c, d e and this may not be a comparator very different kind of inputs. Then if you want to kind of check you know something has gone wrong at 300nanosecond.

And you want to check and modify something so, you go to different part of the code. You know like at 300 for a 300 for b maybe another place 300 for c. So, it is a very comparison code. Because the stimulus is part f the code and it is all over, you know it is kind of distributed at many places and to verify what are the inputs you have supplied what is if you want to modify it is going to be quite tough.

But maybe some simple ways that we could a better method at least in this kind of type of test bench is that instead of giving a sequence of input to a alone what to do is that at 0nanosecond you specify what is a, what is b then give a delay. Then you specify what is a, what is b for 100nanosecond and so on okay. So, that is what is shown a better would be you saying that a gets 0000 ; b gets 0000 then you say wait for 100nanosecond.

So, you can imagine that is much better because if you have c, d, e we will specify everything in one line give a delay. And abcd here give a delay and so on so, all the inputs are the instance a particular instant all the inputs are in one place. And you can easily modify it verified and you can you know append wait and so on. So, this is the better way then this particular thing yet it is still kind of distributed in the code all the way you have to write it manually.

So, it will be good if we can store all the inputs in one place okay. Now in this test bench we are verifying the output manually, we are kind of supplying the inputs. And we are verifying the output waveform okay so, the question is can we in this case say store a and b and for 0000. And we also store say a equal to b, a less than b, a greater than b for this value that means here a equal to b will be 1 other 2 will be 0s okay.

So, you store along with this particular as set off inputs and then what you do is that you apply the inputs stimulus and check that these outputs are equal to what is specified, what is stood along with the inputs. You know so, that is a better method of doing it. So, you do not have to manually verify, if everything goes well you know if there is an error you have to see the waveform what is going wrong.

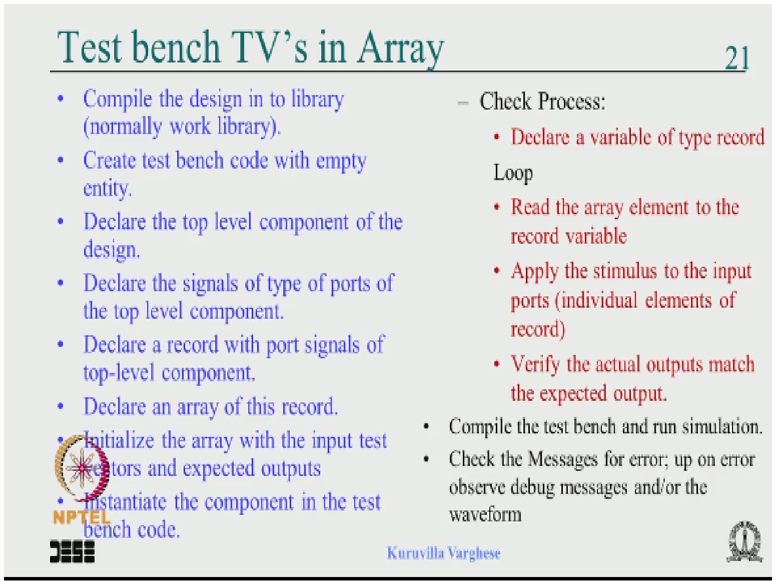
So, that is the better method of doing it so, the question is that can we store input test vectors and expected output together okay. Now mind you that these input test vectors could be of different type. You know there could be 1 which is a single bit another could be a 5 bit vector and the output could be 3 bit vector and 2 bit vector and so on. So, some of you have to combine the data types which are of different types together okay.

So, what is the data structure oh sorry what is the data type that is the best for this to combine together like in our case. You have a, b which are 4 bit vector and then a greater, a equal, a less b is kind of 1 bit standard logic so, which is a data type best for this in a c it is structure and in VHDL it is the record and now record itself is not enough. Because we want to store a sequence of inputs and expected response okay.

So, now but the basic element is record we need an array of record to be able to store different sets you know the inputs expected results at 0nanosecond and inputs expected results at 100nanosecond and so on okay. So, we need a record for individual element and then array of record for sequence of these data okay. So, that is what we are going to do next, we are going to store inputs.

And the expected outputs sequence of m in a array of a record and we are going to extract ii in 1 by 1 apply the input and verify the output again the stored output okay. That is the basic idea.

(Refer Slide Time: 21:50)



Test bench TV's in Array

21

- Compile the design in to library (normally work library).
- Create test bench code with empty entity.
- Declare the top level component of the design.
- Declare the signals of type of ports of the top level component.
- Declare a record with port signals of top-level component.
- Declare an array of this record.
- Initialize the array with the input test vectors and expected outputs
- Instantiate the component in the test bench code.
  - Check Process:
    - Declare a variable of type record

    Loop
    - Read the array element to the record variable
    - Apply the stimulus to the input ports (individual elements of record)
    - Verify the actual outputs match the expected output.
- Compile the test bench and run simulation.
- Check the Messages for error; up on error observe debug messages and/or the waveform

Kuruvilla Varghese

So, let us look at the process which is little more complex than the earlier one so, we compile a design into library and we create a test bench code with the empty entity all same. We declare the top level component it is required same, declare the signals of type of ports of the top level component again this is required because we need to connect a wire when we instantiate. But here is the kind of difference now.

We declare a record with ports signals of top level components these ports we put together in a record, we declare a record of that type with the port signals now we declare an array of this record. We will declare a generic array with unlimited length, now we initialise a look up table of this type array okay. So, with input test vectors and expected output now after that we instantiate the component, then the test bench code.

Now we need to loop through this array read each record by record and apply the inputs. So, that we need a process we declare a variable of type record because we are going to read from the array the 1 by 1, the elements of the array. And now see each element is a record we have to declare a variable of that particular type of record which we have declared earlier. Now in a loop from the beginning of the array to the end.

We **sa** read the current element, array element into record variable because it is an array of record. So, we read the current index record into this particular variable okay. Now that variable allow all the inputs, all the expected outputs so, now we will apply the stimulus to the input port. So, individual elements of the record has to be kind of applied to the input. Then similarly we can verify the actual output again expected output okay.

Now once you do that you can compile the test bench and run simulation and everything is automated you do not have to manually verify the output if everything goes well. If the simulation console say our message saying that test benches completed and there are no errors reported in the console then you are assured that the you design as pass this particular test vectors for verification okay to save.

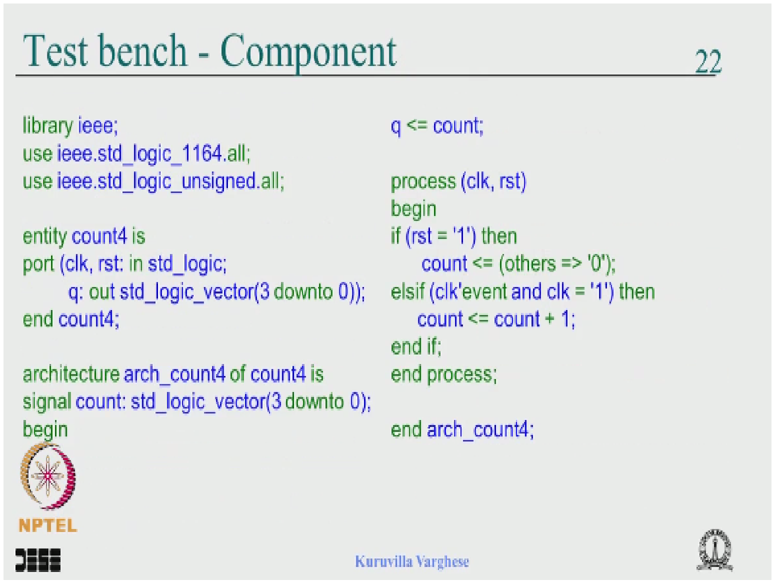
Whether the design is working completely depend on the quality of the test vectors you are applying whether you are you know applying all cases or corner cases which kind of corner cases would mean the extreme cases which kind of typically represent the various behaviour. Suppose the you are designing the ripple adder. So, naturally if you check that the old stages ripple through like say.

You given input all the once one input and the other input is one, then it will be kind of rippling through all the stages. Then you know that all the stages are okay and you might give all 0s all that. You know say kind of over flow hen all 0s depending on the case you have to apply the mind and find the corner cases. You know may not be easier at all and then you have to verify that everything works.

Then you have sure that the design is completely verified so, I suppose that is clear to your mind so, we have compile the design test bench code with the empty entity declare the top level component of the design, declare the signals, declare a record with ports of the top level component, declare an array of record initialise the a kind of look up table with of these type array.

Then instantiate the component in a process you declare the variable of type, the loop through the array read the current elements into this particular variable apply the individual inputs verify the outputs again this stood output that is the process.

(Refer Slide Time: 26:26)



Test bench - Component

22

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity count4 is
port (clk, rst: in std_logic;
      q: out std_logic_vector(3 downto 0));
end count4;

architecture arch_count4 of count4 is
signal count: std_logic_vector(3 downto 0);
begin
q <= count;

process (clk, rst)
begin
if (rst = '1') then
    count <= (others => '0');
elsif (clk'event and clk = '1') then
    count <= count + 1;
end if;
end process;

end arch_count4;
```

NPTEL

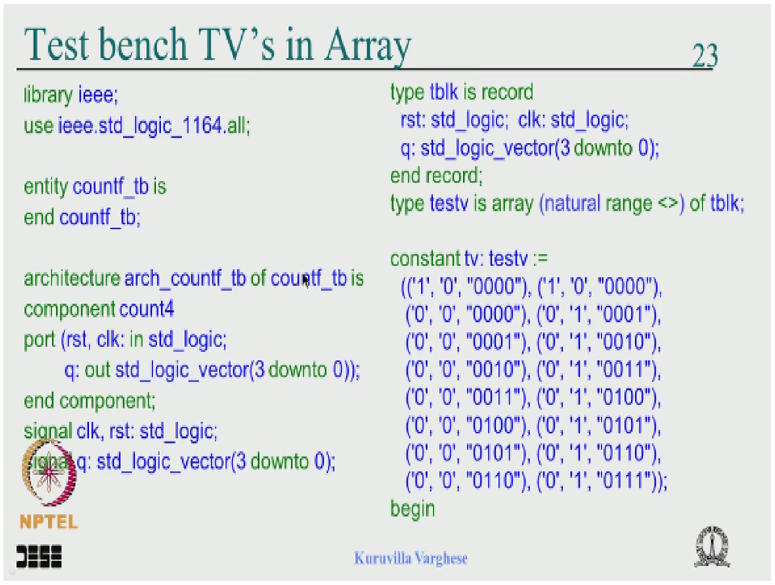
Kuruvilla Varghese

So, let us look at a component and the test bench code the component we are going to use is the 4 bit counter nothing but it is a simple counter. You know you have clock reset and the output okay so, this is the library design, I mean library declaration library package 1164 package unsigned. Because I am going to use a + here so, that is why unsigned is use entity count for is clock reset is in, q is out standard logic vector 3 down to 0.

And now since q is out we cannot write q is q+1 so, we declare a signal count which is standard logic vector 3 down to 0. We write the code in terms of count and you see count is assigned to q so, in the begin we assign the count q we write a process clock reset begin. If reset is 1, count is 0 else if clock event clock is equal to 1 count gets clont+1 okay. So, end if end process so, that is a counter code which we have seen okay.

So, we have an entity a signal to circumvent the kind of buffer type of mode. And then q is assigned to count upon the reset the count is 0, upon the clock the count is count+1 an the process ends and that is the component which we compile into work library. And now we are going to write a test bench using the array to verify the functionality of this particular counter okay.

(Refer Slide Time: 28:00)



Test bench TV's in Array

23

```
library ieee;
use ieee.std_logic_1164.all;

entity countf_tb is
end countf_tb;

architecture arch_countf_tb of countf_tb is
component count4
port (rst, clk: in std_logic;
      q: out std_logic_vector(3 downto 0));
end component;
signal clk, rst: std_logic;
signal q: std_logic_vector(3 downto 0);

type tblk is record
  rst: std_logic;  clk: std_logic;
  q: std_logic_vector(3 downto 0);
end record;
type testv is array (natural range <>) of tblk;

constant tv: testv :=
  (('1', '0', "0000"), ('1', '0', "0000"),
   ('0', '0', "0000"), ('0', '1', "0001"),
   ('0', '0', "0001"), ('0', '1', "0010"),
   ('0', '0', "0010"), ('0', '1', "0011"),
   ('0', '0', "0011"), ('0', '1', "0100"),
   ('0', '0', "0100"), ('0', '1', "0101"),
   ('0', '0', "0101"), ('0', '1', "0110"),
   ('0', '0', "0110"), ('0', '1', "0111"));
begin
```

Kuruvilla Varghese

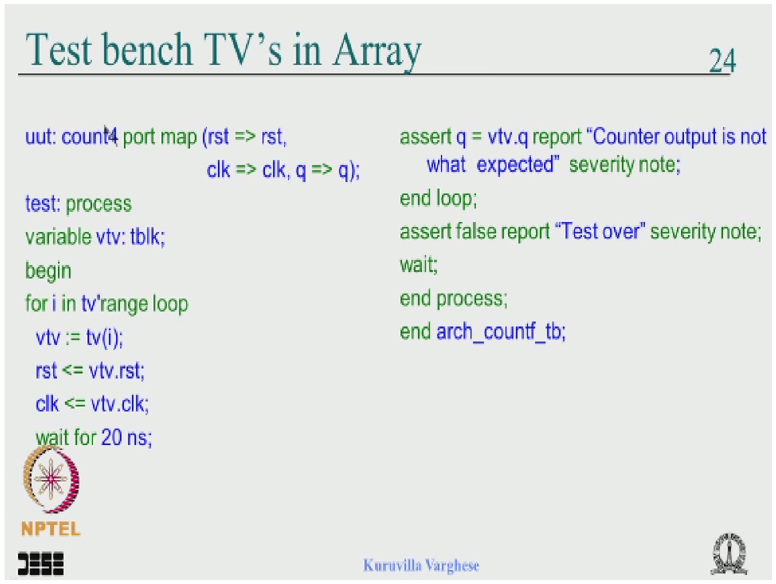
So, let us come to the test bench code initial part looks same we have the library declaration empty entity in the architecture declaration region before the begin. We have component declaration with port reset and clock which is input the q which is 4 bit output, n component. Then we say signal clock reset of this type signal q of this type so, we can instantiate this now before doing going to instantiation we have to declare the record.

So, we say type some name t block is record reset clock and q mind you have shown that in a different order than this okay this particular yeah order is same but it is does not matter even if the order is different it really does not matter. Now we declare an array of the record so, type testv is array instead of giving a particular range. We are giving an unlimited range of the natural numbers natural range unlimited of tblock.

So, testv when we say it is an array of this particular record okay and this is not we are not defining is just a declaration okay. So, this is just the declaration saying that when you tblock it means that, when you say that testv it is an array of this tblock that is the meaning of it. Now we have going to initialise a look up table we say constant because it is 1 time initialise tv, tv is the real look up table. We are going to initialise e colon equal sorry tv is colon.

Then is type this testv which an array of record then we initialise colon equal, then element by element so, the first time the reset is 1 clock is 0. So, the outputs are 0 reset is 1 0 output is 0, then the reset is made 0 output is still 0. Then we give a 1 to a clock so, which is a positive edge count become 1, then we make it 0 the count is still 1 make it 1, then the count is 1 sorry 2 and so on okay so, you make the clock 0, clock 1 and so on okay. You can write as many as you like then you say begin okay.

(Refer Slide Time: 30:42)



Test bench TV's in Array

24

```
uut: count4 port map (rst => rst,
                       clk => clk, q => q);
test: process
variable vtv: tblk;
begin
for i in tv'range loop
  vtv := tv(i);
  rst <= vtv.rst;
  clk <= vtv.clk;
  wait for 20 ns;
assert q = vtv.q report "Counter output is not
   what  expected" severity note;
end loop;
assert false report "Test over" severity note;
wait;
end process;
end arch_countf_tb;
```

NPTEL

Kuruvilla Varghese

Now we instantiate the component count4 is port map reset to reset, clock to clock, q to q, simple now this is the main process which tested. So, test process now mind there is no sensitivity less. Because we are not waiting for an event we are applying inputs okay our self so, there no sensitivity less. So, we declare a variable of this particular record okay to be able to read 1 by 1 so, variable vtv is of type tblock.

Now we begin we loop through our look up table for i in tvtick range okay so, this is an attribute when you say this is tv when you tv tic range it means the it is kind of 0 to whatever number. You know automatically that correct number will come tv sorry for i in tv tic range loop, now vtv that is this variable equal to tv(i) so, that is the current index when it is 0. The zeroth index this particular one is red into this particular variable.

You know this all 3 are red into this vtv now we have to apply the reset and clock. We say reset get vtv dor reset individual element, clock gets vtv clock. So, then we give a delay wait for 20nanosecond now this is the game we say assert that means you make sure that q that is the actual output of the counter is equal to vtv . q. So, that is this stored value like this particular one for the time instance 0. So, vtv. Q. So, the syntax of assert is that.

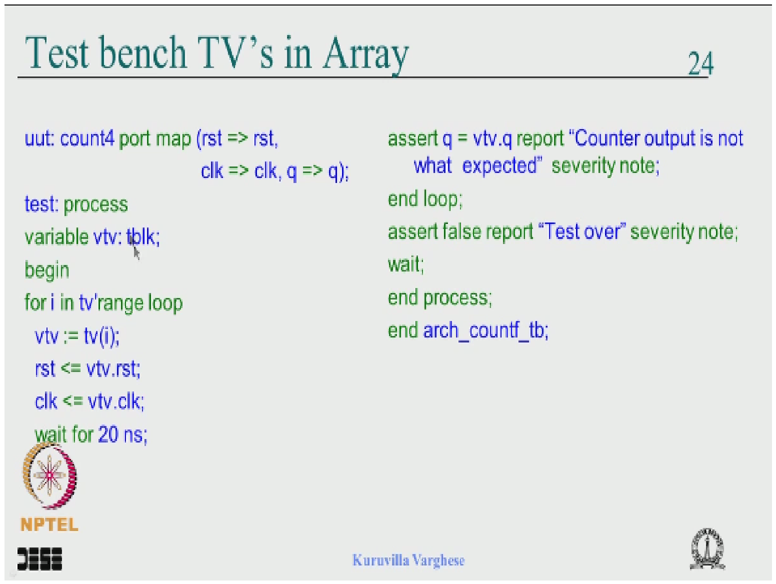
If it is true keep quiet, if it is false then you report you know whatever is t written message written will come into the console will be printed on the console. So, if it is not equal then the console will say 0nanosecond counter output is not what is expected. Then you can go and verified and there are very severities different severities. And you say severity note and warning means it will just printed.

If you say error or failure it will exit this simulation okay so, normally it is enough if you use auto warning. So, you say end loop okay so, this is the loop which loop through look up table apply the inputs 1 by 1 verify the outputs at the end. We say assert false report test over severity notes, when you say false this will be printed any way severity note there are you know later VHDL syntaxes allows you to write without assert false.

You just a report some message and will come on the screen but do not take risk. Because maybe that is part of the VHDL 2012 which may not be all 1997 which may not be supported in some kind of some simulator simulators sometime allows you to select particular language version. You have supposed to use that all that but very important, you know that the process does not have a sensitivity less. So, we cannot just say n process, then again it will go back to the process.

So we just say wait, you know everything is over, you wait okay otherwise it will go on doing it. Because we are writing all these in the concurrent board, you have the VHDL code so, this is always active. So, if there is no sensitivity less, then t will go through after getting out of the loop will start again. So, that is avoided by same wait and then n process so, that is how we store that test vectors in an array with expected result okay. So, once again quick run library package the empty entity component declaration, signal declaration record declaration, array of record declaration. Then initialise the look up table.

(Refer Slide Time: 35:20)



Test bench TV's in Array

```
uut: count4 port map (rst => rst,
                      clk => clk, q => q);
test: process
variable vtv: tblk;
begin
for i in tv'range loop
  vtv := tv(i);
  rst <= vtv.rst;
  clk <= vtv.clk;
  wait for 20 ns;
assert q = vtv.q report "Counter output is not
   what  expected"  severity note;
end loop;
assert false report "Test over" severity note;
wait;
end process;
end arch_countf_tb;
```

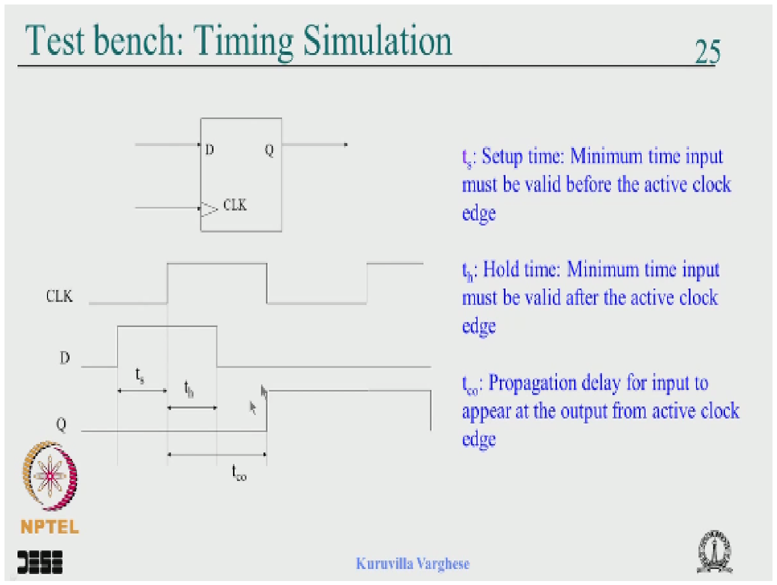
NPTEL
Kuruvilla Varghese

Then you we begin instantiate the component in the test process we declare the variable then loop through the look up table read the current index into the variable apply the individual elements to the input give a delay check that actual output is equal to the stored output then if you not you write a message the end of the loop when you come out of the loop you flash a message to the screen, then you wait and end process.

So, that is how the test bench you know this is a real very useful test bench okay now only problem with this test benches that this is for functional simulation okay. Now if you write this kind of test bench it cannot be used in timing simulation okay. That you have to be very careful because even if you take a simple combinational circuit of course the counter is the sequential circuit you know that you apply the input.

Output will be available only after some delay, when you do timing simulation so, if you just apply the input and given a arbitrary delay. And if you check the output, output may not be ready at all. So, maybe the total delay is 25nanosecond. So, if you apply the input wait for 20nanosecond. And check if the actual output is equal to the expected output. It will show error okay so, when you do timing simulation you have to make sure that. All the time parameters specify are correct and in a case of simple combination circuit this is the propagation delay.

(Refer Slide Time: 37:05)



Test bench: Timing Simulation

t_s: Setup time: Minimum time input must be valid before the active clock edge

t_h: Hold time: Minimum time input must be valid after the active clock edge

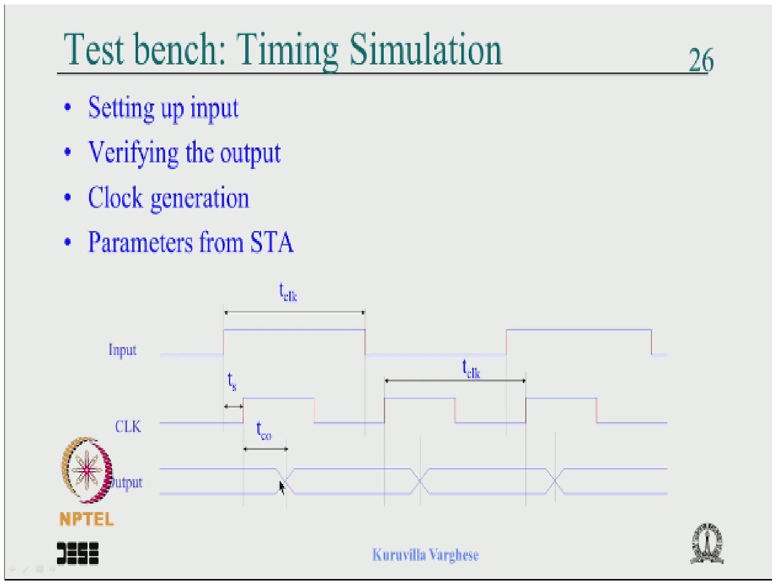
t_{co}: Propagation delay for input to appear at the output from active clock edge

Kuruvilla Varghese

But when we come to sequential circuit it is not only verifying the output even applying the input, we have to meet some timing requirement okay. So, in a sequential circuit you know that the input should applied to the flip-flop sometime before the clock edge okay. So, in a test bench when you apply the input you have to make sure that it is applied sometime before the setup time with respective to active clock edge.

Similarly the input should remain there for some more time hold time. Now once that is done we have sure that the output will be input will be transferred to the output. But that we can check only after the propagation delay tco of the flip-flops. So, that also we need to ensure. One is meeting the setup and hold time, another is verifying the result after the worst case the tco okay.

(Refer Slide Time: 38:10)



Test bench: Timing Simulation

- Setting up input
- Verifying the output
- Clock generation
- Parameters from STA

So, that I am showing in a waveform and it means that we have to be careful when you apply the input we have to be careful when you verify the proper output at the proper instances we have to do this, we have to generate a clock and let us see suppose we generate a clock in the simulator periodic clock okay. Now our job is that always come before the setup time with respect to active clock edge apply the input.

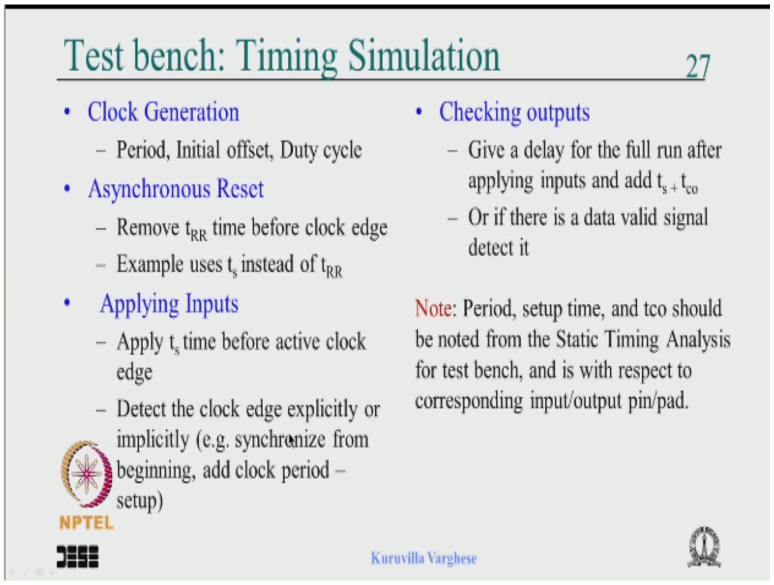
Now the moment you are say setup time before this clock edge what you need to do for applying the next input is just wait for a clock period. Then correctly you have before setup time, before the next active clock edge. So, once you are correctly behind the clock edge by setup time. it is enough if you keep on adding 1 clock period apply the input, 1 clock period apply the inputs and so on okay.

Now verifying the output you have to like after the applying the input. Then you have to wait for the clock edge. Then wait for tco then you verify that, the output is equal to stored output okay. So, that is one way of doing is that first up all you come setup time before the clock edge you know that if you detect the clock edge there is no way to go back. But what you can do is that suppose you have detected the clock edge by clock tic event clock is equal to 1 or at the beginning you know that it is starting with a half clock period from 0.

Then you can wait for half clock period–setup okay or you somehow latch on to active clock edge by implicitly or explicitly. Then you can say wait for a clock period that is from here to here–setup time then you are correctly behind that okay. Now once you are there then the next clock edge, the input will be latch and the output can be tested say from this is you say + setup time + tco then you are here.

Then you can verify the output now to come here you can say a period–tco, because we were in like so much in front of tco by tco you say form here is a +period – tco –setup then you have correctly here okay. And we will not in this game we will not store the clock not required clock is a periodical waveform. So, we can write a loop to generate the clock okay. So, that is have the timing simulation is done.

(Refer Slide Time: 40:57)



Test bench: Timing Simulation

27

- Clock Generation
  - Period, Initial offset, Duty cycle
- Asynchronous Reset
  - Remove t_{RR} time before clock edge
  - Example uses t_s instead of t_{RR}
- Applying Inputs
  - Apply t_s time before active clock edge
  - Detect the clock edge explicitly or implicitly (e.g. synchronize from beginning, add clock period – setup)
- Checking outputs
  - Give a delay for the full run after applying inputs and add $t_{s+}t_{co}$
  - Or if there is a data valid signal detect it

Note: Period, setup time, and tco should be noted from the Static Timing Analysis for test bench, and is with respect to corresponding input/output pin/pad.

Kuruvilla Varghese

So, I will kind of illustrate various part of it one is generating the clock. So, in a loop we will generate the clock in a process and we can specify the period we can specify the duty cycle if it is not square wave and we can give an initial offset you know maybe for 100nanosecond clock can be inactive say that might kind of reflect real life scenario, because there could be PLL base clock managers and which will at the beginning it will take some time for pl the lock.

So, we can to simulate that we can kind of given initial offset or for any other purpose and very important thing to remember is that when you apply asynchronous reset okay. You can apply the reset any kind. Because it has priority you know just apply anywhere the output will go to 0 irrespective of the clock. But when you remove the reset if it is near to the clock edge. Then that can create metastability .

We have not studied it, but then essentially the synchronous reset what it is doing is that is going into manipulate the internal latches directly okay. Now the problem is that if you are removing the input in this setup time into very near to the clock edge the input is also manipulating the active latch and now there will be kind of contention and by the time you remove the reset there may not be enough time for the input to drive the internal latch.

And that can be it metastability. So, when you apply the input you remove the input when you apply the reset remove the reset sometime before the clock edge, it is called reset recovery time fo safety you know in this code I am assuming that I am removing it by the setup time okay. Before the clock edge by setup time. now very important question to ask is that where do you get this information.

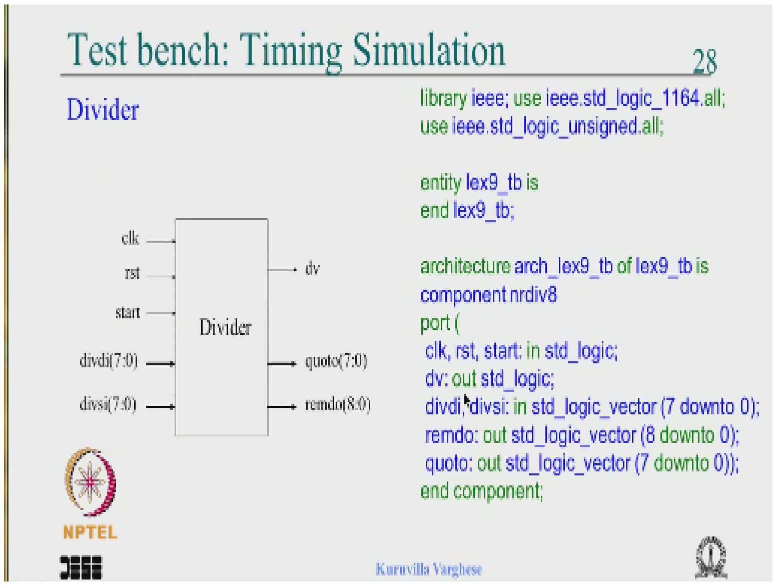
We are not even simulated how do we know what is the case to be applied or what is the tco to be taken or what is the clock period to be taken now this will be part of the static timing analysis had briefly mention that static timing analysis do the timing estimation by looking at the delay of the block, delay of the wires okay. It can go wrong but you can specify the cases where you the tool should not analyse like false path multi cycle path.

And all that so, if you do that the static time analysis will give correct estimate then you can use that plugged in the test bench to verify write a verification code with respect to timing okay. So, that is what is asynchronous reset apply the reset before the set up time, then applying the input is that we detect the clock edge either you say clock event clock is equal to 1 or you synchronise from the beginning.

Because we know that then the process start it is at 0nanosecond and we know how the clock is going at 0nanosecond to the period by 2 it is 0. Then the clock goes for half period and all that so, we can make out what the going on and can be in synchrony with the clock generation and we say a you can wait for a period – set up time. Then it will be write behind the clock edge by set up time okay.

Now once you give a delay of the full run maybe the design you are doing it will take multiple clock cycles. You can give wait for all that clock cycle and check whether data is valid and this I have taken kind of after Xilinx test bench template. And this is how they do it and I have used that template particular template.

(Refer Slide Time: 44:56)



Test bench: Timing Simulation

28

Divider

clk
rst
start
divdi(7:0)
divsi(7:0)
Divider
dv
quoto(7:0)
remdo(8:0)

```
library ieee; use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity lex9_tb is
end lex9_tb;

architecture arch_lex9_tb of lex9_tb is
component nrdiv8
port (
 clk, rst, start: in std_logic;
 dv: out std_logic;
 divdi, divsi: in std_logic_vector (7 downto 0);
 remdo: out std_logic_vector (8 downto 0);
 quoto: out std_logic_vector (7 downto 0));
end component;
```

NPTEL

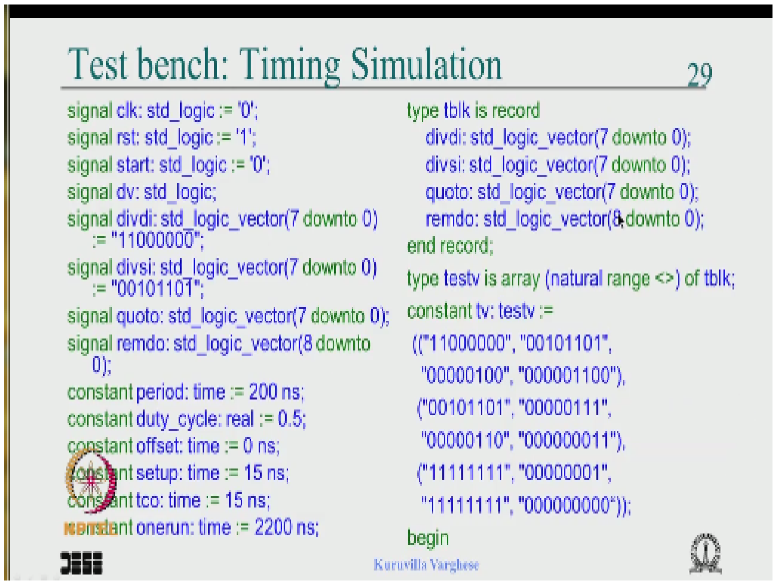
Kuruvilla Varghese

And for a illustration so, component we are going to check the functionalities is a divider 8 bit divider. So, you can see there is a clock and reset because it is a complete complex divider complex in the sense at it is a multiple cycle divider it has start signal. You apply the input dividend and deviser which are 8 bit, then you give a start pulse. It will start dividing do the divider operation.

Then quotient and remainder will come here and a data valued signal will indicate that the data is valid. So, you know that division is over okay so, that is the component clock reset start and dividend deviser quotient and remainder. And dv is indicating that the division is over and what we do is that we generate a clock in a process. We give reset and start manually because it is only required reset is required only at beginning a the power on.

Start is required only one spare of every iteration okay so, we will not store all these values in the array. We will store dividend deviser quotient and remainder in an array. And in a loop we will apply the input give a start signal, then remove the start signal wait for this data valid or wait for the complete run. Then verify the output okay that is the game. And let us see the code we have the library and package code. I am using unsigned because some where I am going to use.

(Refer Slide Time: 46:34)



Test bench: Timing Simulation

29

```
signal clk: std_logic := '0';
signal rst: std_logic := '1';
signal start: std_logic := '0';
signal dv: std_logic;
signal divdi: std_logic_vector(7 downto 0)
    := "11000000";
signal divsi: std_logic_vector(7 downto 0)
    := "00101101";
signal quoto: std_logic_vector(7 downto 0);
signal remdo: std_logic_vector(8 downto
    0);
constant period: time := 200 ns;
constant duty_cycle: real := 0.5;
constant offset: time := 0 ns;
constant setup: time := 15 ns;
constant tco: time := 15 ns;
constant onerun: time := 2200 ns;
```

```
type tblk is record
   divdi: std_logic_vector(7 downto 0);
   divsi: std_logic_vector(7 downto 0);
   quoto: std_logic_vector(7 downto 0);
   remdo: std_logic_vector(8 downto 0);
end record;
type testv is array (natural range <>) of tblk;
constant tv: testv :=
 (("11000000", "00101101",
   "00000100", "000001100"),
  ("00101101", "00000111",
   "00000110", "000000011"),
  ("11111111", "00000001",
   "11111111", "000000000"));
begin
```

Kuruvilla Varghese

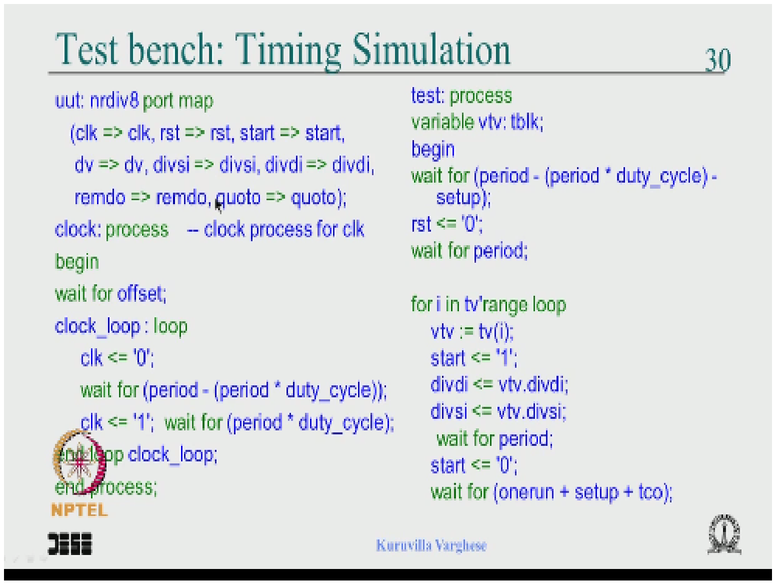
That is why to use and empty entity and this is in the architecture declaration region, we have the component which is called nrddiv8 is 8 bit divider. So I indicate nr by nrd storing you do not worry about it. It something some case study which have done so, we will just take a it is just name and the port is clock reset start dv dividend deviser remainder and quotient so, these are inputs and these two are output, dv is output.

So, you declare the component then we declare the signals and mind you can see that reset is applied at the 1 at the beginning okay. Clock is made 0 at the beginning, start is made 0 so, all these are made to some kind of useful values at the beginning itself okay. Start i s0 because we make it inactive reset is 1. Because at beginning itself it is reset and dividend and deviser also we giving some time some input.

So, that you do not see all the red waveforms otherwise it will show some kind of u or a z or some u it looks not good to have a simulation having lot of red waveforms. So, this avoids that and the quotient and remainder now these are the constant which is use for clock generation and the timing so, the period is 200nanosecond, duty cycle is 0.5 square wave initial offset is 0nanosecond. So, you asked us not having offset set up time is 15nanosecond, tco is 15.

One run is 2200nanosecond okay so, that is around 11 clock cycles of 200nanosecond okay and these are picked up from period set up tco is picker up picked up from the static timing analysis. So, we declare the record which is of dividend deviser quotient and remainder declare and array of a this particular record. Then we a look up table constant tv is testv, then we say 4 values dividend deviser quotient remainder and so on okay. So, however many values you can write not a problem.

(Refer Slide Time: 49:01)



Test bench: Timing Simulation

30

```
uut: nrdiv8 port map
  (clk => clk, rst => rst, start => start,
   dv => dv, divsi => divsi, divdi => divdi,
   remdo => remdo, quoto => quoto);
clock: process   -- clock process for clk
begin
wait for offset;
clock_loop : loop
    clk <= '0';
    wait for (period - (period * duty_cycle));
    clk <= '1';  wait for (period * duty_cycle);
end loop clock_loop;
end process;
```

```
test: process
variable vtv: tblk;
begin
wait for (period - (period * duty_cycle) -
    setup);
rst <= '0';
wait for period;

for i in tv'range loop
   vtv := tv(i);
   start <= '1';
   divdi <= vtv.divdi;
   divsi <= vtv.divsi;
    wait for period;
   start <= '0';
   wait for (onerun + setup + tco);
```

Kuruvilla Varghese

Then we really started then we have the component instantiate of the nrd8 component and this is a position named association formal clock is map to actual clock reset to reset start to start and so on okay. So, all the inputs and outputs are map to appropriate signal now we write a process for the clock. You say clock process no sensitivity less so, it is going to loop forever, you say begin and then wait for offset some initial delay.

Then we say we write a loop now which is clock loop we say loop so, you say infinite loop it is not a say for i in 0 to something it is just goes on looping yet clock is initially made 0 wait for period–period in duty cycle okay. Because you know we start with 0, then goes 1 for a duty cycle okay so, that is period into duty cycle so, we have to say total period –period into duty cycle that is the duration of the 0.

Then you say make it 1, then wait for period into duty cycle. So, you see now it add ups period–period into duty cycle+period into duty cycle is the period. So, is a n loop clock loops so, it keeps on looping and keep on generating clock cycle one after the other. So, remember that the clock is 0 for the period–period into duty cycle one for period into duty cycle okay in our case it is 50 % so, clock is 0 at the beginning from 0 to 100nanosecond.

Because it is 200nanosecond, then from 100nanosecond, 200nanosecond it is 1and it repeats okay. Now this is the test process similar to the previous one. So, we write a process without sensitivity less you say variable, vtv is t block. Now we say now this is where the differences earlier we just apply the signal. But now initially we made reset is 1, now divider as the reset internal registers are reset, internal state machine is reset.

Now we say we remove the reset, now we said avoid metastability we have to remove sometime before and we have taken it as setup time. so, you say wait for period, so the 1 full clock period –period into duty cycle–setup okay. So, we are just coming. So, because it is you know it is kind of if you see this is where we are the beginning, because it is 0 then 1 we are saying wait for 1 clock period–period into duty cycles.

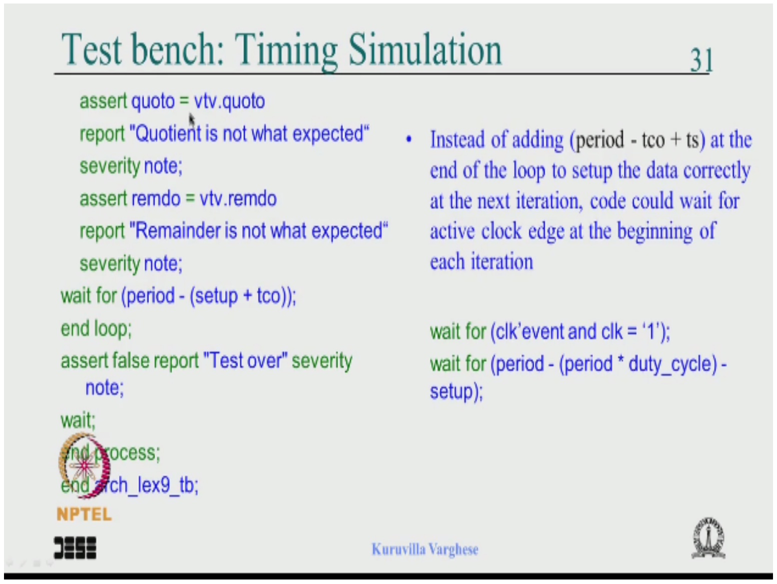
So, we are here–setup time, so we are here okay. So, that is what is shown here, wait for period–period into duty cycle–setup, now we remove the reset correctly to avoid reset to give reset recovery time, reset is 0. Now we are correctly before the setup time, before the clock edge you say wait for period. So, we are at right point now we say for i in tv' range loop we are looping through this particular look up table, we say we read the first element vtv; equal to tv(i) okay.

Now what we do is that we have to apply the input give a start okay. So, we say start is 1dividend is vtv. Dividend whatever was red, deviser is vtv.divisi input okay. That we are reading from this and this okay, now input is apply we can we have to remove the start signal. Because if start signal is 1, it will end and again restart the internal divider state machine will restart again.

So, we give only the start for a clock period and it is correctly applied before the setup time all these 3 wait for a period. So, we are still before the setup time it make start 0, so correctly we have applied the inputs applied the start, remove the start at the correct page. Now the divider will start dividing and so we say wait for 1 run that is a time taken for complete division or you can write it as you know so many clock cycles into the clock period+setup time.

Because we are behind setup time before the active clock edge+setup time+tco, because we have going to check the output, so +tco.

(Refer Slide Time: 54:03)



Test bench: Timing Simulation

31

```
  assert quoto = vtv.quoto
  report "Quotient is not what expected"
  severity note;
  assert remdo = vtv.remdo
  report "Remainder is not what expected"
  severity note;
wait for (period - (setup + tco));
end loop;
assert false report "Test over" severity
   note;
wait;
end process;
end arch_lex9_tb;
```

- Instead of adding (period - tco + ts) at the end of the loop to setup the data correctly at the next iteration, code could wait for active clock edge at the beginning of each iteration

```
wait for (clk'event and clk = '1');
wait for (period - (period * duty_cycle) -
setup);
```

Kuruvilla Varghese

Now we verify the quotient and the reminder we say assert or quotient is equal to store quotient report if it is not correct. Similarly assert reminder is vtv. reminder, otherwise report reminder is not what is expected and we say severity note okay. So, that is how we verify the quotient and reminder. Now we were now here you know at this point of the simulation we are at this point, we have verifying the output.

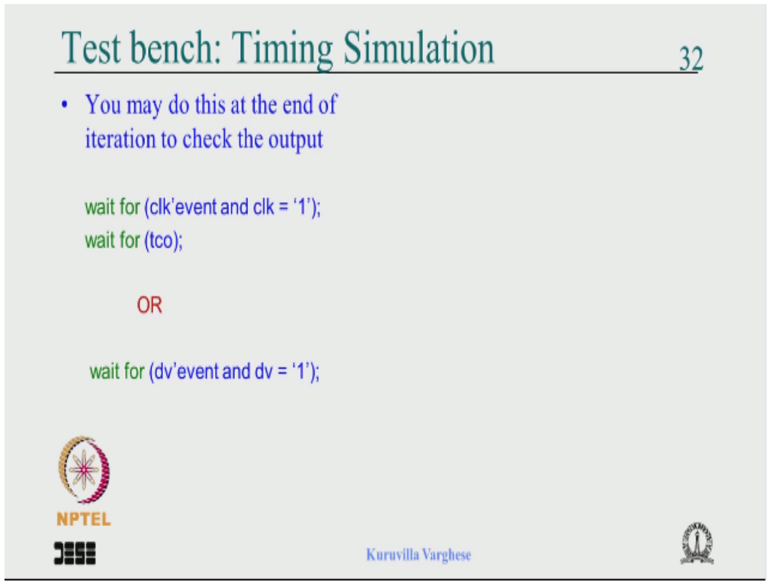
Now before the next clock cycle we have to come here. So, what we do is that we say wait for a 1 full clock period–tco–setup time okay. So, we are write here okay. So, that is what we have doing here we say wait for a period–setup+tco or–setup–tco. And then it goes back you know the old process is repeated then again you apply the input give the start signal, remove the start signal, verify the output and so on.

End loop, so that is where everything is applied say assert false report, test over, severity note you know that is the game is over. So, 1 quick run through this we have empty entity, component declaration signal declaration with default values. These are parameters for clock generation and timing record, array of record look at table, instantiate the component, clock process with a loop for you know.

In finite loop for generating the clock, test process declare the variable come before the setup time, remove the reset loop through the array read the current element apply the start, apply the input wait for a period remove the start, then wait for the complete run add setup time tco, verify the output, then before coming to the next iteration come behind the setup time, then loop print report okay.

Now there is another way of doing this explicitly you can say for setup time you can say wait for clock tic event clock is equal to 1. Then you are on the clock edge, then you can say wait for period–period into duty cycle for you know meeting the setup times.

(Refer Slide Time: 56:32)



Test bench: Timing Simulation

32

- You may do this at the end of iteration to check the output

```
wait for (clk'event and clk = '1');
wait for (tco);

        OR

wait for (dv'event and dv = '1');
```

NPTEL

Kuruvilla Varghese

Similarly for checking the output you can say wait for clock event clock is equal to 1 at some after a 1 run or something like that and you say wait for tco or if there is data valid you can say wait for dv event dv=1 okay. So, there are different method of doing it, so basically test benches are nothing but is pure VHDL code, you know there is nothing to do with synthesis.

So, you have free to write the way you like I have shown you some kind of probable possible ways of correctly doing it 2 was 1 was very simple, then was an array which is good for functional simulation. But then I have shown real timing simulation, so my advice is that at the beginning itself you write attest bench which is fit for timing simulation. So, you can use that for functional simulation also which avoids you know repeating you know again and again 1 for functional simulation.

One for timing simulation not required write 1 for timing simulation use it for functional simulation with some value it does not matter, but when it comes to timing simulation you have to go to the static timing analysis pickup the worst case setup time hold time sorry. Setup time and the tco and the clock period and do that you know polite that in the timing simulation test bench code.

And then you can run the simulation I will show that when we declare kind of you know if the lecture on the case study and the tool I will show this particular thing no issue. So, please go through it revise understand it properly I wish you all the best and thank you.