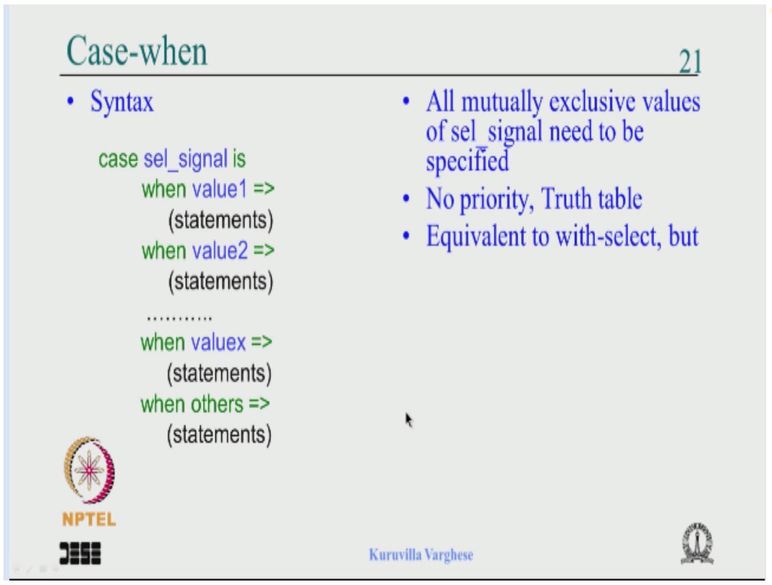
Digital Systems Design with PLDs and FPGAs

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

Lecture-16
Modelling flip-flops, Registers

Welcome to this lecture on VHDL in the digital system designed with PLDs and FPGA course. The last lecture we have seen the sequential construct case 1, and the loops the generate loops and the 4 loops and we have also started looking at the VHDL modelling of the sequential elements like flip-flops latches and so on. So, let us run through the last lectures part quickly. So, let us turn to the slide.

(Refer Slide Time: 01:04)



Case-when

21

- Syntax

```
case sel_signal is
    when value1 =>
        (statements)
    when value2 =>
        (statements)
    ..........
    when valuex =>
        (statements)
    when others =>
        (statements)
```

- All mutually exclusive values of sel_signal need to be specified
- No priority, Truth table
- Equivalent to with-select, but

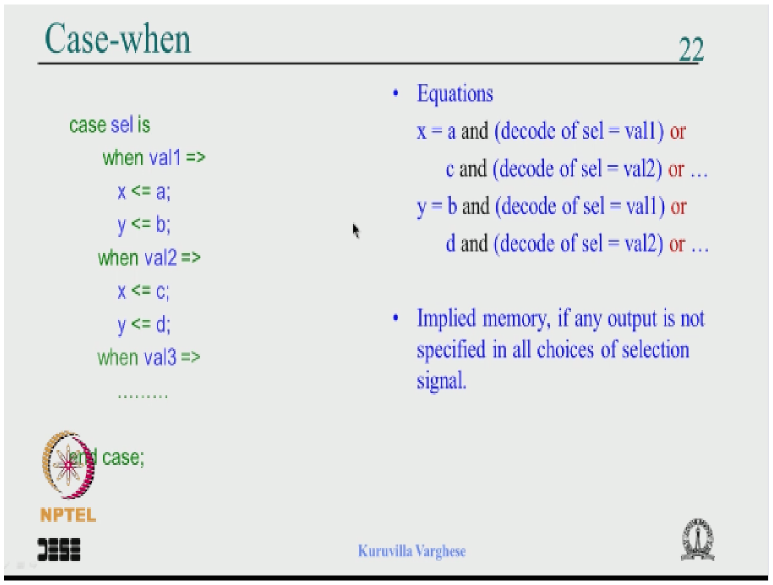
Kuruvilla Varghese

This is the syntax of the case when this is equivalent to the which select of the concurrent statement. The syntax is that case input signal is and when value 1 value of this select signal numerical value. You do the assignment statements okay various appropriate statements can come here, and value 2 at then you say when others which is others means whatever is not included in this is combined together in others.

So, the rule is that mutually exclusive values of select signal all mutually exclusive value should be specified and exactly like which select a it is nothing. But truth table we are specifying the output the either the numerical values are in terms of input for the all possible values of some group of signal or some signal. Because you could if you have three inputs you could even combine them into a single vector.

And write this it is possible to do that it is equivalent to which select but there is a difference you can specify the multiple outputs needs case. You can do the nesting that means a case can be nested within suppose you say when value 1, you can have a another case statement okay or you can write a nest within one of the when you know one of the choice is. So you can have very complex combination circuits specified by this structure the syntax.

(Refer Slide Time: 02:57)



Case-when

22

```
case sel is
    when val1 =>
        x <= a;
        y <= b;
    when val2 =>
        x <= c;
        y <= d;
    when val3 =>
        .........
end case;
```

- Equations

  x = a and (decode of sel = val1) or
  c and (decode of sel = val2) or ...

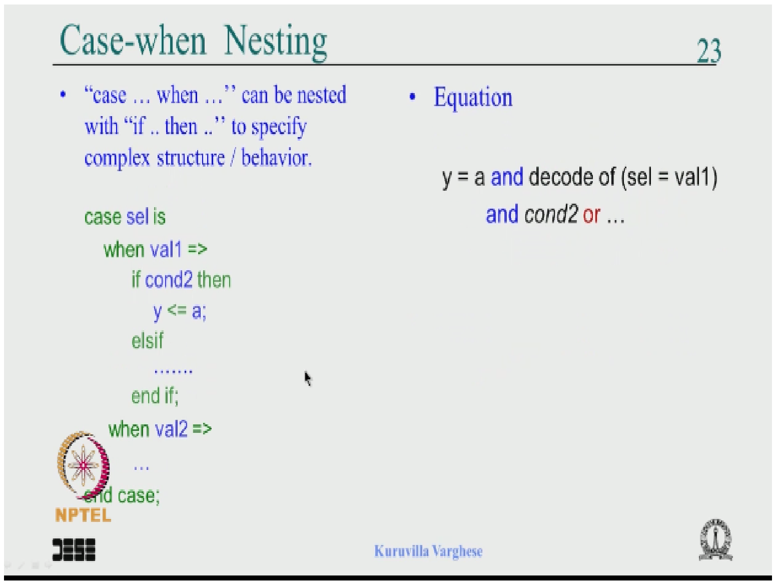
  y = b and (decode of sel = val1) or
  d and (decode of sel = val2) or ...

- Implied memory, if any output is not specified in all choices of selection signal.

Kuruvilla Varghese

We have seen an example suppose we have a therefore value1, value2 if you have a and b specify, then the equation is x is a and decode of value1 or say c and decode of value2 or like that it goes okay. And if you miss anything suppose you fail to specify x here. Then for that value it is Implied latch then will be a feedback so, you need to take care say imagine

(Refer Slide Time: 03:32)



Case-when Nesting

23

- "case … when …" can be nested with "if .. then .." to specify complex structure / behavior.

```
case sel is
  when val1 =>
    if cond2 then
      y <= a;
    elsif
      .......
    end if;
  when val2 =>
    ...
end case;
```

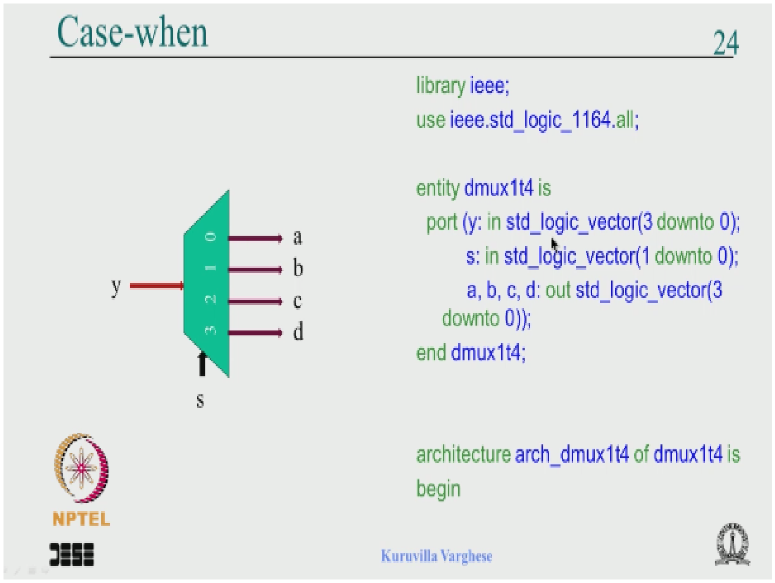
- Equation

y = a and decode of (sel = val1) and *cond2* or …

Kuruvilla Varghese

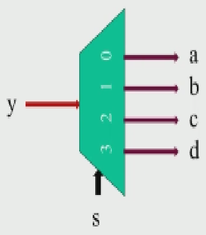
If you could makes case with if or a complex kind of structure so, here I am showing we have a input signal select. So case sel is when value1. I am saying if condition 2 is as some as we discuss it is some Boolean condition composing of some inputs okay. And y gets a else if it goes like that so, if you look at this part. Then the equation is y is a and condition2. And sorry and decode of select equivalent value1 and condition2 as I said it can be composed of the or of many product terms. It expands it is a very kind of little abstract way of saying a that a the equation.

(Refer Slide Time: 04:30)



Case-when

24



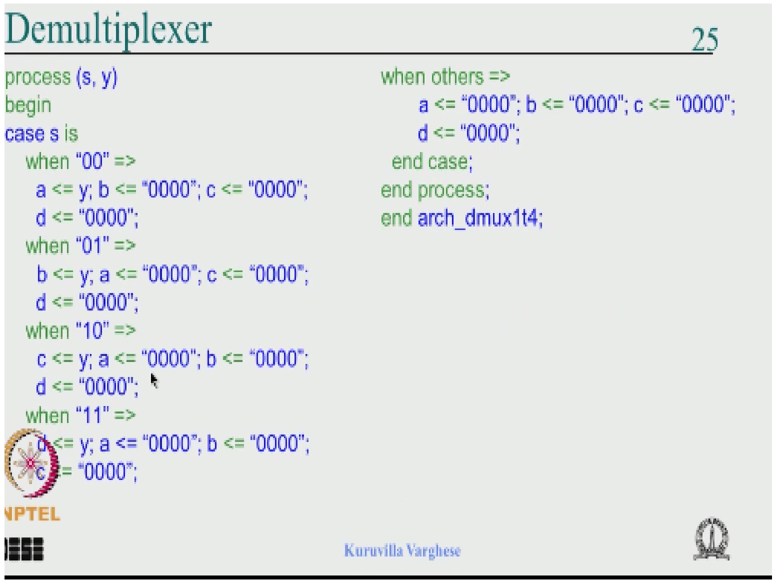
```
library ieee;
use ieee.std_logic_1164.all;

entity dmux1t4 is
  port (y: in std_logic_vector(3 downto 0);
       s: in std_logic_vector(1 downto 0);
       a, b, c, d: out std_logic_vector(3
    downto 0));
end dmux1t4;

architecture arch_dmux1t4 of dmux1t4 is
begin
```

Kuruvilla Varghese

But it can expand and we have seen an example of a case when construct we have taken 1 to 4 de-multiplexer each is for bit. You know the input is for bit outputs are for bit so, if the select line is 2 bits because there of 4 outputs, if it is 0 then y goes to a 1 y goes to b and so on. So that is a library declaration this is a entity where in we have y which is a for bit vector s 2 bit vector and abcd are the outputs which of for bit vector.

And this is a architecture declaration region before the begin we have nothing to declare in the statement region. We are going to use a case when so, this is a sequential syntax so, you have to use a process and in the sensitivity less input should be there. And so, y and s has to be in the sensitivity less, then we are going to use the select line for all the values of select line. We are going to specify the output that is the most natural thing for a multiplexer or a de-multiplexer that is the natural way of describing the behaviour.

(Refer Slide Time: 05:53)



Demultiplexer 25

```
process (s, y)
begin
case s is
  when "00" =>
    a <= y; b <= "0000"; c <= "0000";
    d <= "0000";
  when "01" =>
    b <= y; a <= "0000"; c <= "0000";
    d <= "0000";
  when "10" =>
    c <= y; a <= "0000"; b <= "0000";
    d <= "0000";
  when "11" =>
    d <= y; a <= "0000"; b <= "0000";
    c <= "0000";
when others =>
    a <= "0000"; b <= "0000"; c <= "0000";
    d <= "0000";
  end case;
end process;
end arch_dmux1t4;
```

NPTEL
Kuruvilla Varghese

So, that is shown here process s, y this is select line this is a input begin case s is and when s is 00. We say a gets y and we have to specify b c d everything okay. Similarly for 01 b gets y, and c gets y, d gets y for 11 rest all is 00. And we can say when others everything is 0 you could even write others here. Because this is mainly for simulation, and we can combine 11 with others only thing is that.

If you happen do give some unexpected values like z z to the select line, then it is difficult to debug this 11 with other cases and all that. And one more thing what I have found is that some people some students think inactive means tri-state and the tri-state it. It is a very dangerous thing to tri stated, because you have an output a single output which is just a 1 output which is going to many input.

If you tri-stated it means that this lines of floating it is connected with the high resistant to VDD high resistant of ground. So it is not strong drive the input is kind of floating and the various inputs which is driven by this output can treat it is depending on the some noise pick up. It can start switching okay so, do not tri stated unless it is a bus, and even in the case of bus it

When a everything is tri stated it has to be pulled up or pulled down weakly for a proper state otherwise it is dangerous to have a bus tri-stated particularly an output tri stated. So, please understand that and one problem with this code is that everywhere, you have to write all the outputs it can be kind of composum to tracked and all that. So there is a better way that is at the beginning you make everything 0.

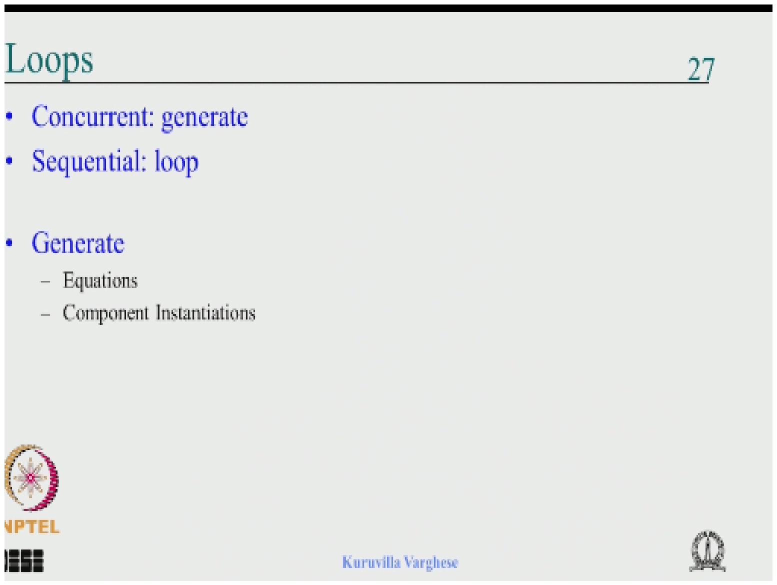
And in choices you make only whatever is required okay. That is no issue because, you know that as for as simulator is concern. When event happens it goes from top to bottom so, a gets a suppose this select line was 00 then the a gets 00. But immediately in the same kind of the process computation a gets y so, that is replace. So, there no at same t + delta a gets y it is correct. And synthesis tool should make out that it is at the beginning it is initialise to 0. So synthesis tool will look at the code part.

And will Inferred this kind of behaviour there should not be a problem. But I said that I warned you that do not take this kind of to an extreme play with delta cycle play with all kinds of initialisation and hoping that. You are reasoning would be kind of understood by synthesis tool, but definitely the tools are getting smarter okay tools are getting smarter means that the tools vendors are kind of accommodating more and more.

Now is engineers or dummies and many a times we are happy that you write some kind of arbitrary code and the tool is making sense out of it okay. That in my opinion is not a very good trend in the sense that because, there is no rules specified you know you do something and you get most of the time things correct. Because the tool vendors no that.

What all kinds of possibilities are the that time to incorporate, but some time reasoning wise it is does not make a much sense. So, do not be very happy that you write something and the tools make sense out of it. I mean better strict to rules and what is most you know kind of reasonable in strict, whether then you know take too much liberty with this kind of syntax. And then we have seen the loops.

(Refer Slide Time: 10:36)



Loops

27

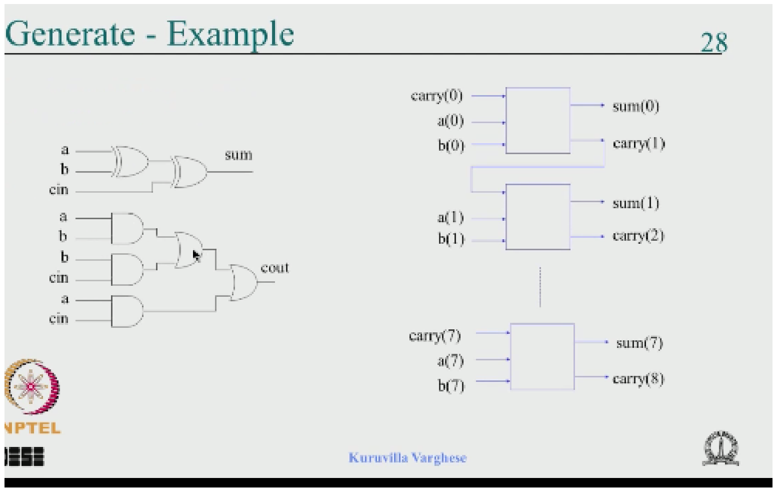
- Concurrent: generate
- Sequential: loop

- Generate
  - Equations
  - Component Instantiations

NPTEL
Kuruvilla Varghese

The concurrent loops are generate and the sequential loop is for loop and generate is use for with equations that means if you have a kind of regular modular structure repeated, in equation or in components that can be easily compress by the generate construct that is a the idea behind it.

(Refer Slide Time: 11:06)



Generate - Example

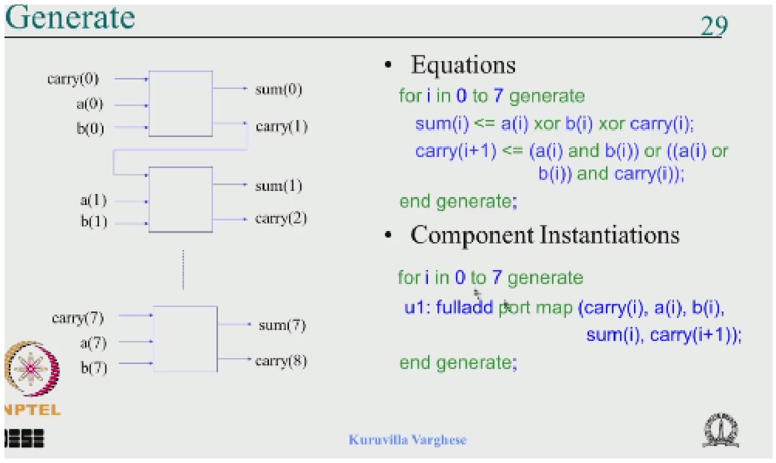
28

And we have seen as an example a ripple adder 8 bit ripple adder here. And you know that basic building block is full adder where in there are inputs are 3 a b and c in 2 outputs are there. The sum is a XOR b XOR c and the carry out is a b or b c or a c okay. That means more than 2 or more inputs are 1. Then the carry is 1, and when we try to make a ripple adder you know that a0 b0 and c0 comes here.

Sum0 is output the output of the first stage is carry 1, which is fed into the carry input of the second state. Then you get sum1 carry2 and it goes on okay. So, if you write an entity you should have 2 inputs which is 8 bit which goes from a7 down to 0, b7 down to 0. These are the inputs and the outputs are you know the sum 7 down to 0 okay. And if you want you can include this carry as part of the sum. You can say sum 8 down to 0.

It does not matter how you define and we need an internal signal maybe this time is mention the last class in at this an internal signal carry which is 8 down to 0. Because we have for convenience we are naming this carry and definitely this carry0 can be if you want a defined as a carry in and is carry 8 can be kind assigned to sum 8 or carry out depending on your need. But that is the how the entity is declared.

(Refer Slide Time: 12:59)



Generate

29

- Equations

```
for i in 0 to 7 generate
   sum(i) <= a(i) xor b(i) xor carry(i);
   carry(i+1) <= (a(i) and b(i)) or ((a(i) or
                 b(i)) and carry(i));
end generate;
```

- Component Instantiations

```
for i in 0 to 7 generate
 u1: fulladd port map (carry(i), a(i), b(i),
                       sum(i), carry(i+1));
end generate;
```

Kuruvilla Varghese

Then you can write equation in generate loop which say for i in 0 to 7 generate say here sum of i is a of i XOR b of XOR carry i. So, it is sum0 is a0, b0, carry0. And carry i + 1 is ai bi or ai or bi and ci. This is nothing but a ai ci and bi ci okay that when you expand and here you see that when for the zeroth loop these are 0s and this is 1. So you get a carry1 as a output in the next iteration for the sum the carry1 is a input.

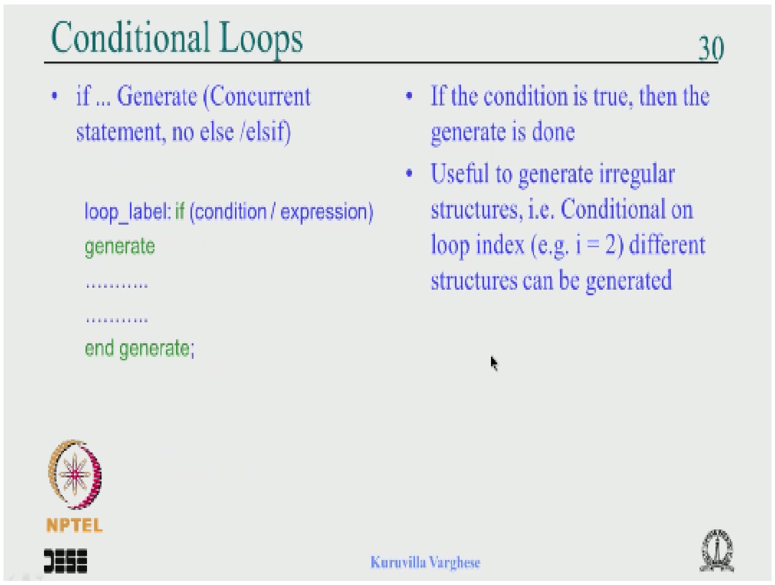
So naturally this happens okay now with regard to component instantiation you write an entity and architecture to implement this. But now you write it top level code, I suggest, you write the code for it with component declared as a full adder of this particular full adder. We have written then you define a signal called carry which is a you know 9 bit then you do this kind of component instantiation through generate loop for i in 0 to 7 generate.

Then give a label you say full add that is a the entity name of this port map and I am assuming it is in ports are define in this order carry a b sum and carry out. So, you say carry i ai bi sum i in carry i + 1. So this also naturally it will happen the zeroth iteration it is carry0 a0 b0. Sum0 is output carry1 is output and the next iteration comes the carry1 is input. So this output of the first stage goes as a input of the second stage and you get the structure by this simple statement.

And as I said this can be use very easily when the structure is very symmetric of balance or regular. But if you have an regular structure like say you imagine carry look at adder. It is little more complex the first stage is small the second is stage is bigger than the first one. It have more inputs and the third stage will how much more inputs like second stage will have you know the not only a1 b1 it will have a0 b0 as a input.

And carry0 so, as it comes down it expands and it is little tough to write generate loop for that it is maybe not possible also. But you can write a 4 loop which is sequential we will see that maybe or you try to from whatever I say I suggest you work it out. You try to make an 8 bit carry look at adder using either the generate loop or the sequential loop. You might need little more input on that maybe I will give as we go along.

(Refer Slide Time: 16:13)



Conditional Loops

30

- if ... Generate (Concurrent statement, no else /elsif)

```
loop_label: if (condition / expression)
generate
...........
...........
end generate;
```

- If the condition is true, then the generate is done
- Useful to generate irregular structures, i.e. Conditional on loop index (e.g. i = 2) different structures can be generated

NPTEL
Kuruvilla Varghese

And you can have conditional generate as some condition if some condition is true, then you can say generate this is true for generating some regular structures like as I said maybe carry look at adder, you have to try I am not tried it because once again one thing connected with that I want to make a mention many a times the hardware, when we write design.

We do not worry about the length of the code okay like in a software you write say c code then you know that if somebody has you have an algorithm you are trying to implement somebody has implemented that algorithm in say 200 lines. And somebody else has written the same algorithm in 100 lines say without loops and all that I am not talking about loops.

Then it is for sure that c code with 100 lines will execute faster complete faster then the c code with the 200 lines. But in when you talk about hardware this need not to be true because, the hardware is not trying to execute your line it is going to synthesis some circuit out of the written code. So, it may happen that somebody write and hardware description of a circuit which is longer than somebody else.

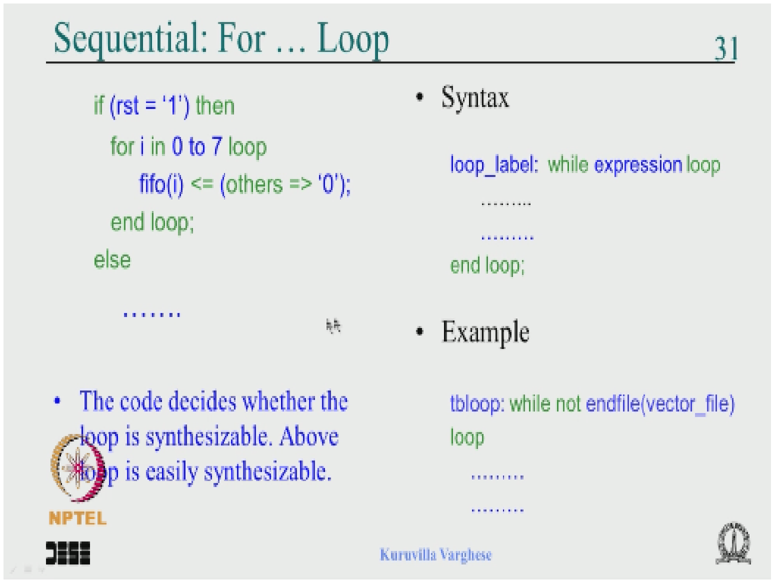
But it might turn out that this, whatever is written longer will produce a small of faster circuit it is possible okay me for an example. But then if you care you can work it out okay there no issue. So, when you are I mean your designing hardware using hardware description language. You have worried need not be the number of lines of codes okay, so maybe because you know that.

We many a times work with 8 bits, 16 bit, 32 bit 64 bit even if you need to copy paste something modify something sometimes it is not worthwhile to try to write very concise loops which can work for, you know the 8 bit, 16 bit, 32 bit, 64 bit and 1024 bit got knows when a 128 bit processor comes or it is required are the such memories sizes are possible.

It may come but then it is not a great need to write a loops and compressed a code and sometime may compressing, you could write a very complex loop to come out with the irregular structure you know you can take great pain to make it very generic and all that. But sometime the readability goes down it sometime it is worth the effect. You know how much time it takes a suppose you take an 8 bit the carry looker adder as an example**.**

But maybe in like in VLSI you do not look use carry look at adder, but then it is not a great effort copy paste and modify it might take 10 minutes to correctly code it does not really it is in matter. So I just mention this because it is maybe it is a right place should mention it that is all.

(Refer Slide Time: 20:08)



Sequential: For … Loop

31

```
if (rst = '1') then
   for i in 0 to 7 loop
      fifo(i) <= (others => '0');
   end loop;
else
   .......
```

- The code decides whether the loop is synthesizable. Above loop is easily synthesizable.

- Syntax

```
loop_label: while expression loop
   .........
   .........
end loop;
```

- Example

```
tbloop: while not endfile(vector_file)
loop
   .........
   .........
```

Kuruvilla Varghese

So let us come to back with. So, sequential loop the syntax is for i in something 0 to 7 loop instead of generate n loop and you write statement here okay. That is the syntax. And here I am showing a for loop like it say is reset is 1 for i in 0 to 7 FIFO i is others 0 n loop okay. There are other ways of a writing this you could write without a loop. If you want that is not what a but it is an example and this quite synthesisable.

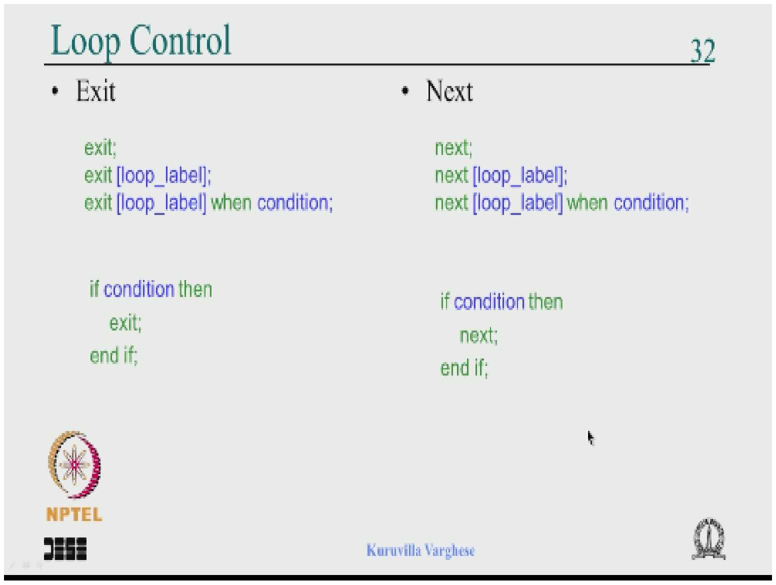
There no issue because the as I said the FIFO each location is initialise to 0. And this is 8 bit location and FIFO as maybe so, many entries maybe 256 entries or a 2k entries it does not matter. It essentially means that the all the flip-flops used in FIFO the reset is connected together and when it is asserted it becomes 0. That nothing it is not a great it is easy to synthesis.

And people make very generic statement like loop is not synthesisable, which is not true which as I said it has essentially means that it you are an algorithm where loop is there if you directly translate that into VHDL you may not get the circuit which is implements that algorithm. That is a meaning when somebody is say a makes a sweeping statement the problem is that many a times.

The speakers themselves does not understand what they are saying what they are probably they are repeating something they have heard. So, that is all so, do not get worry it too much about it and here also in the loop also you have conditional loops. You can say while some expression is true. Then you can loop n loop and one example is that in the case of test bench. You would have written test vectors in a file, and normally you read line by line.

And the top level code will be like this there outer loop will be like this while not a end file vector file and this end file is a procedure or a function which say when you reach the end of the file. Then loop if you have not reach the end of the vector file keep reading the lines and do whatever is required. That is the meaning of it that is one place it is quite useful not for synthesis this is useful in simulation for verification.

(Refer Slide Time: 22:47)


Loop Control

32

- Exit

```
exit;
exit [loop_label];
exit [loop_label] when condition;

if condition then
   exit;
end if;
```

- Next

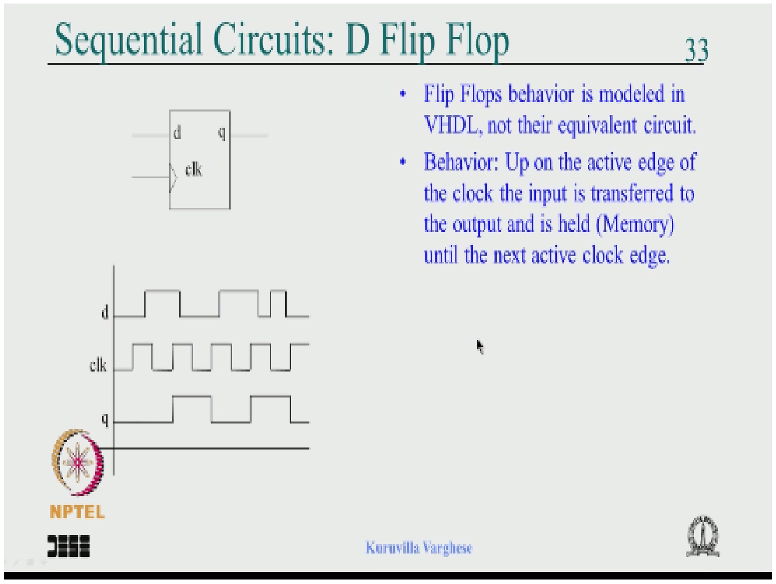
```
next;
next [loop_label];
next [loop_label] when condition;

if condition then
   next;
end if;
```

NPTEL

Kuruvilla Varghese

You have loop control the exit that means wherever you say exit can be exited not very useful you know kind of the plain exit, because that means you are not using that loop after that when you encounter exit. But you can say on a condition when some condition is met ,you can exit as I said you can make some regular structures or you can write if condition1 then exit. Next is skipping the loop again a conditional shipping maybe useful. It is similar to the sequential language you could even write instead of next when condition, you could write if condition then next end if.

(Refer Slide Time: 23:36)


Sequential Circuits: D Flip Flop

33

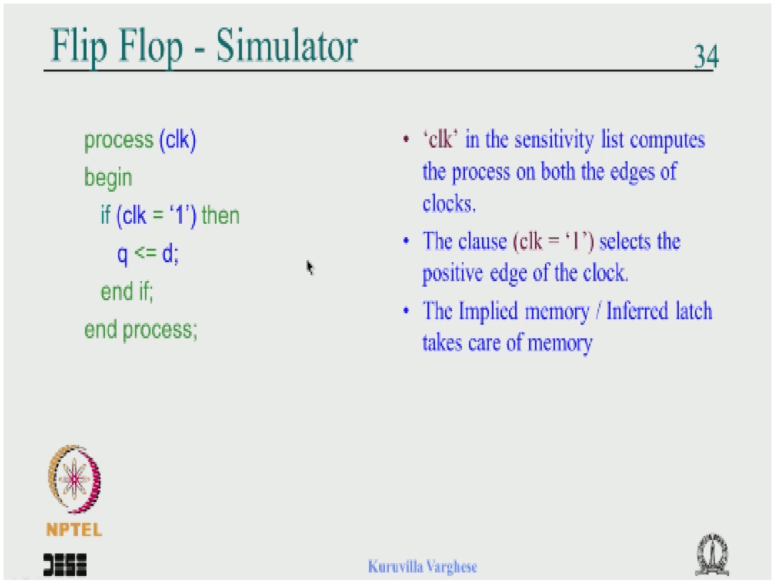
- Flip Flops behavior is modeled in VHDL, not their equivalent circuit.
- Behavior: Up on the active edge of the clock the input is transferred to the output and is held (Memory) until the next active clock edge.

NPTEL

Kuruvilla Varghese

And we have looked at the sequential circuit elements the modelling and we looked at the flip-flop behaviour is that if a clock comes active clock comes the d is transferred to the q.

(Refer Slide Time: 23:53)



Flip Flop - Simulator

34

```
process (clk)
begin
  if (clk = '1') then
    q <= d;
  end if;
end process;
```

- 'clk' in the sensitivity list computes the process on both the edges of clocks.
- The clause (clk = '1') selects the positive edge of the clock.
- The Implied memory / Inferred latch takes care of memory

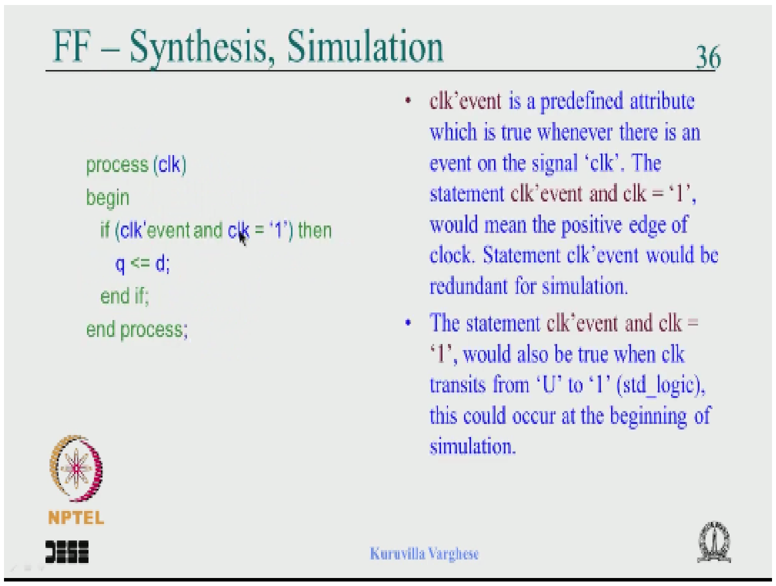
Kuruvilla Varghese

And we have written we have written a process with the clock in the sensitivity less this you know kind of catch the edge the transition any event happens on this. And in the code you write if clock is equal to 1 along with if this event and this clock is equal to 1 will make it kind of the positive edge. Then you say q equal to d and you say end if that means if this condition is not met you remember that.

You know that is it is a perfect code for simulator. But as I said the synthesis tool look at the code within the process body, and that says if clock is 1 q gets b. That means as long as the clock is 1 q is d and it will become a transparent latch okay. And, this end if is the one which is specifying memory element. So, we are using that Implied memory to kind of the describe the behaviour of the flip-flop.

So, there is a compatibility problem with this code like this works this is a flip-flop for simulator. But it is a latch for synthesis tool so, essentially to write the flip-flops for you know the synthesis tool you need to bring in the event on the clock.

(Refer Slide Time: 25:24)



FF – Synthesis, Simulation

36

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q <= d;
  end if;
end process;
```

- clk'event is a predefined attribute which is true whenever there is an event on the signal 'clk'. The statement clk'event and clk = '1', would mean the positive edge of clock. Statement clk'event would be redundant for simulation.
- The statement clk'event and clk = '1', would also be true when clk transits from 'U' to '1' (std_logic), this could occur at the beginning of simulation.

NPTEL

Kuruvilla Varghese

And that is by specifying like this, you say a clock tick event that is mean any when on the clock and clock is equal to 1, q gets d end if but the problem for the simulator is that there is a redundancy like if this says that if there is a event on the clock. But once again it comes to the code that is repeated, but it is still okay because we cannot do without this sensitivity less. So, for the synthesis tool.

We write clock tick event and clock is equal to 1 and this kind of takes care of the event on the clock as for as synthesis tool is concern for simulator. There is a redundancy that the simulator starts computation when there is an event on the clock and once again it is repeated. But it is okay because this is a code where there is a compatibility between synthesis and simulation.

Earlier code is dangerous in the sense that it works as a flip-flop for simulation and latch for synthesis. This works as a flip-flop for both simulation and synthesis there are two things to remember that when you say clock tick event and clock is equal to 1. So we are kind of reasoning out the behaviour saying that it means the positive edge okay.

So, now you if you think you know if you under round the syntax of VHDL and comes it some other ways of describing this particular condition. It may not work with the synthesis tool, because this is accepted as a template for the positive edge okay. So, it is not that everything every possible way, you try to represent this kind of behaviour. This synthesis tool will understand.

Because that is a just software even if you apply kind of machine learning to it there is less possibility that a so, you should think that you are when you say synthesis tool it is just a set of algorithms software which try to make sense out of all the code you write. So, you should stick to the standard rules of the game not try to invent a kind of engineers ways of describing the behaviour.

And hoping that will be kind of accepted as that particular behaviour okay. And another problem here is that when you say clock tic event it could mean any event it is not that 0 to 1. Because we use standard logic and you know that an example is that when you start simulation normally everything is kind of initialise as you if it is not initialise properly so, it may happen that the clock is flagged as you and when you start applying the clock.

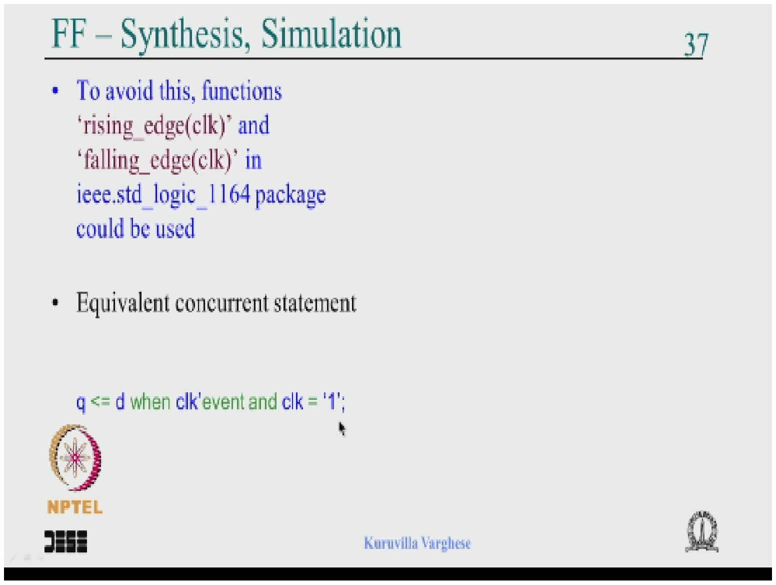
It becomes transit 1 or 0 such a thing and happen then this can be is a valued condition as a kind of it is a u to 1 and 1 then this can get latch. That means there is no problem in the real hardware it does not happen in the real life. But in simulation this can happen and that can be problematic, because suppose you have a counter which is counting something.

And if this happens it will start counting at the beginning only there is a miscount at the beginning. But when you in real life you look at it that the count will be one less so, this you should be very careful there no harm writing the code like that. But when you simulate make sure that the behaviour of the simulation and behaviour of the real circuit is identical and particularly with initialisation and miss trigger and all that.

So, that you get what you are hoping and what you are expecting and you do not kind of otherwise maybe you are hoping that you know the behaviour properly, but you do not know why that happens in simulation you do something else to sort it out without understanding the problem then that will work may not work with the real life circuit so, always keep very you know alert on the initial part of the simulation.

All initialisation should be proper particularly with this respect to this kind of at data types it initial values the code you written all that. So, but there is a way to overcome that the problem is with this attribute clock tick event is an predefined attribute which only means an event on clock okay. It is not that it is 0 to 1 or 1 to 0.

(Refer Slide Time: 30:42)



FF – Synthesis, Simulation

37

- To avoid this, functions 'rising_edge(clk)' and 'falling_edge(clk)' in ieee.std_logic_1164 package could be used
- Equivalent concurrent statement

```
q <= d when clk'event and clk = '1';
```

NPTEL

Kuruvilla Varghese

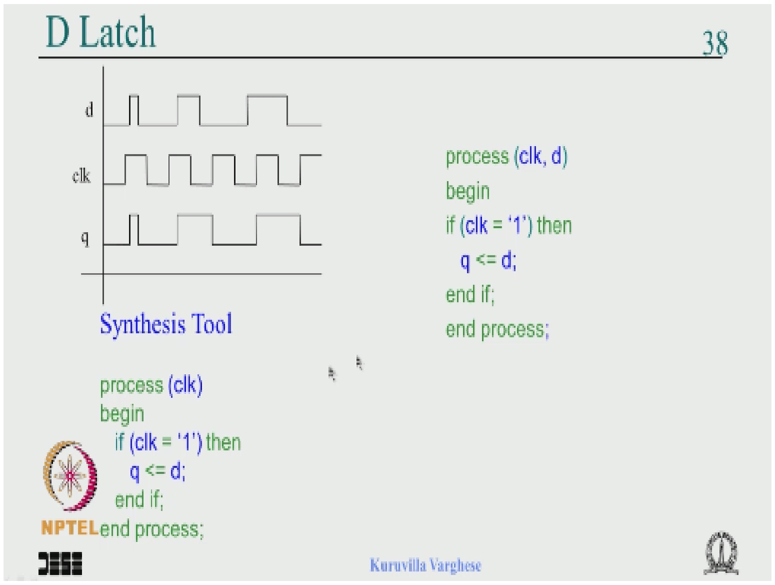
But way to avoid that is that there is a function in standard logic 1164 package. There are 2 function one is called rising edge in the bracket clock and falling edge bracket clock. This correctly you know returns Boolean true if there is a transition on clock from 0 to 1, similarly falling edge will return it true. If there is a falling edge are negative edge transition on the clock only problem is that and you write.

You know instead of if clock event clock is equal to 1. You write if rising edge clock and it is a the return value of that function is the Boolean true or false. So if rising edge clock you can write the only problem is that in terms of computation each times such a thing encounters this particular function is called so, if you take a maybe some 1% or 2% of the computation time could be extra may not be big deal with the percent.

They kind of computing power that is only kind of negative or the small part of it. And the question is that can be have the concurrent statement which specify which by which you can write this code the flip-flop okay. So, this says that if clock tic event and clock is equal to 1, then q gets d end if okay so, naturally you know that when else is equivalent to if so, if you try to do this.

You can say q gets d when clock tic event and clock is equal to 1. And you know that when as an else part you do not say that okay. Then you get the equivalent concurrent statement so, q gets d when clock tic event and clock is equal to 1. Then you get the flip-flops using concurrent statement at least the simple flip-flops with the concurrent statement.

(Refer Slide Time: 33:07)



D Latch

38

Synthesis Tool

```
process (clk)
begin
  if (clk = '1') then
    q <= d;
  end if;
end process;
```

```
process (clk, d)
begin
if (clk = '1') then
   q <= d;
end if;
end process;
```

NPTEL

Kuruvilla Varghese

So, let us come to the d latch now the earlier we wrote a code for simulator the flip-flops code for simulator. Then it happened to be a latch for synthesis tool so, let us put the let us look at the latch functionality. So, it is that if clock is 1 then whatever is at the d will come on to q, when the clock goes low it remembers okay. So, the code as for as the synthesis tool is concern. You say process clock begin if clock is 1 then q gets d end if.

The synthesis tool looks only at this if clock is 1, then q gets d end if so, this part is correct you know. But for simulator see if you write only the clock in the sensitivity less. So, this condition is checked only there is an event on the clock okay. There is an event either goes from low to high or high to low, then the clock is 1 is checked and q is assigned d.

But you know that when the clock is 1 if d changes this code will not make the q d as for as simulator is concern. So, basically because it is not sensitive to d, so for the simulator to get a latch, you have to make the input d in the sensitivity less. So, that is a code for the correct code for latch as for as simulator is concern for synthesis tool absolutely no problem, because it looks at this code.

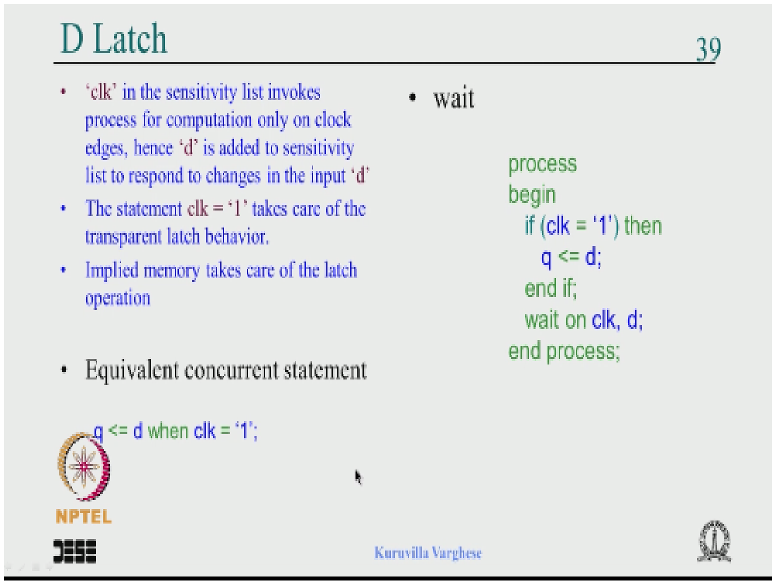
It perfectly okay whatever you write here so, that is a code process clock d begin if clock is 1 then q gets d end if it works because if there is an event on the clock. So, it comes and check clock is 1 then q is reflected after sometime the clock remains same. But there is a change in d so, that comes here, if clock is 1 q gets d. But if clock is 0 suppose there is an event on d and clock is 0 like this you know.

There is a event on d clock is 0 nothing happens it remembers because event on d it is return clock is 1 give q gets d. If clock is 0 nothing happens because of end if it retains a the old value so, this is the code for transparent latch which is compatible both for simulator and the synthesis tool so, we have looked at the latch code and the flip-flop code, there are some details.

You need to understand so, I suggest that you know grasp this concept clearly so, that you clear because once you have clear, because we are kind of come to the core part of the VHDL as for as synthesis is concern. So, we are kind of completing we have completed the sequential sorry the combination circuit perfectly because, you know how to what is the meaning of, which select, when else if then case, when loops all that you know.

So, as for as combination circuit is concern and now we are in sequential circuit main thing is the registers and the combination circuit. So, you we have learn how the combination circuit can be coded. Now we have we are learning how the registers of flip-flops can be coded. So, this together should work only think is that maybe what need to be learn is that maybe can we combine this together. So, that you get less code or something like that. That we will see. But we are kind of coming to a as for as synthesis is concern, we are dealing with the most of it. So, please understand the concept well so, that you can design properly.

(Refer Slide Time: 37:32)


D Latch

- 'clk' in the sensitivity list invokes process for computation only on clock edges, hence 'd' is added to sensitivity list to respond to changes in the input 'd'
- The statement clk = '1' takes care of the transparent latch behavior.
- Implied memory takes care of the latch operation
- Equivalent concurrent statement

```
q <= d when clk = '1';
```

- wait

```
process
begin
  if (clk = '1') then
    q <= d;
  end if;
  wait on clk, d;
end process;
```

NPTEL

Kuruvilla Varghese

So, here the question is that can we have an concurrent statement equivalent to this like here the again if the main cracks of the coding is if clock is 1 q gets d end if okay. So, you can say q gets d when clock is 1 and terminate it, you do not say end if so, that gives an Implied latch and it works perfectly. So, equivalent concurrent statement is that q gets d when clock is 1 okay.

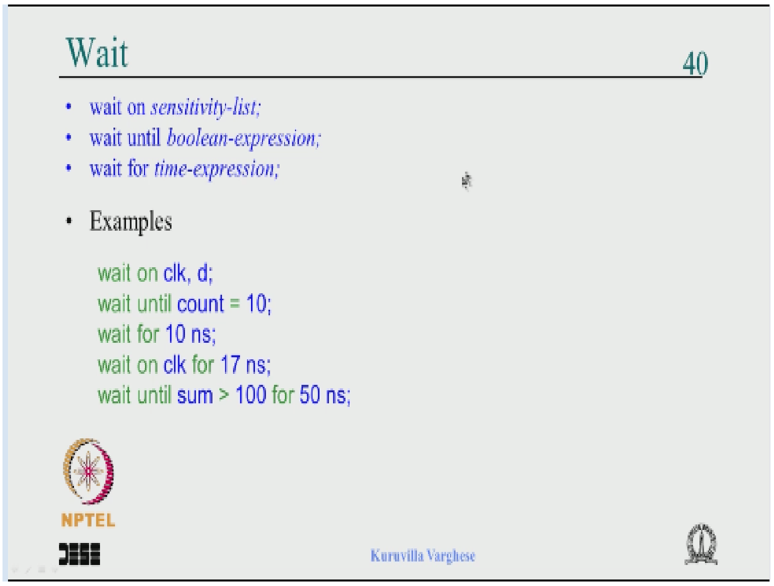
That means whenever there is an event on the and you know that the as for as the concurrent statement is concern the simulator computes whenever there is an event on the right hand side. So, if there is an event on d even on clock, this will be kind of computed so, which say that if there is an event on d when on clock then this will be computed that say if clock is 1 q gets d. Otherwise if clock is 0 it remembers.

So, this works perfectly with regard to this and there is another syntax like we have looked at the this particular syntax as for as latches concern. But there is a another way of writing it instead of writing the signals in the sensitivity less, you could write use a wait statement so, that is what is shown here. Please look at the code and try to make a sense out of it. So, in this code you have no sensitivity less.

And you say process without any qualification and here you write the code and you say wait on clock and d okay. So, normally when you have a process the wait goes is that at the beginning it computes once. Then it waits for event on the kind of the signals. So, it is similar like at the beginning it is computed 1. Then it is waiting on for any event on clock and if. If there is an event on clock and d goes and compute once from top to bottom.

And again wait for event. It is nothing it exactly identical, either you specify the clock and d here, or at the end you say wait on whatever was here. Mind you should not do both. Then it does not work, because there is a it waits for an kind of event on this, computes again it waits for event on that and event happens it comes back and gets struck there. So, either you should use wait statement or sensitivity less not both. So, let us maybe it is a right time to look at the kind of the syntax of this wait statement.

(Refer Slide Time: 40:40)



Wait

40

- wait on *sensitivity-list;*
- wait until *boolean-expression;*
- wait for *time-expression;*

- Examples

```
wait on clk, d;
wait until count = 10;
wait for 10 ns;
wait on clk for 17 ns;
wait until sum > 100 for 50 ns;
```

NPTEL

Kuruvilla Varghese

And there are 3 kind of types of wait, you can ask free looked at before you can say wait on a list of signal, sensitivity less. You can say wait until some condition is true okay or you can say wait for some expression in terms of time you know. You can say wait for 10 nanosecond okay like that. So, this is what we have seen wait on we have seen, you say wait on clock and d. That means wait for some event on clock and d.

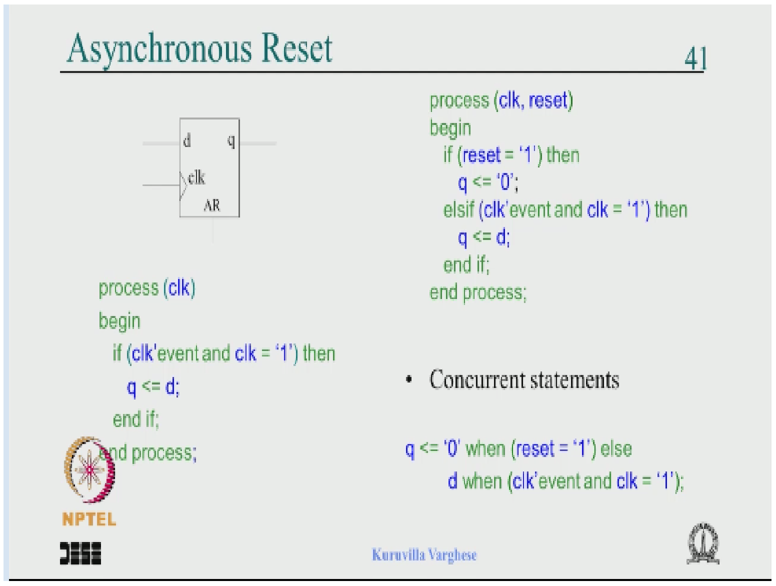
It can be use for both for synthesis and simulation no issue. Some as part of the simulation you want to like you are writing a test bench, you have waiting for some event on a clock to continue with a test bench you can do that. And you can write this wait until count is 10. That means you come to some place. You have some counting happens. You can say wait until count is 10 okay.

Absolutely no problem with the simulation and for synthesis you have to think okay. If in the code you can really make sense that it can easily for yourself work out the structure for synthesis then it works. Otherwise be very careful. And wait for 10 nanosecond it is only for simulation. Wherever you say it wait so, just do not think that you are assign some output and you say wait for 10 nanosecond.

The synthesis tool will give you a delay 10 nanosecond or something like that. And you can combine these wait on and wait until with the 4 and things like that here. Wait on clock for 17 nanosecond okay, can be use for simulation. It means that wait for an event on clock or 17 nanosecond. I mean for 17 nanosecond you wait on clock, I mean at 10 nanosecond if there is an event on the clock continue.

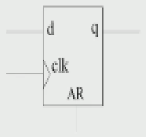
Otherwise till 17 nanosecond the simulator waits for an event, otherwise it continues. Exactly same like this you have say wait until some is greater than 100 for 50 nanosecond. You say that wait for some to be greater than 100. But that you do wait for such an event only for 50 nanosecond. If that does not happen for 50 nanosecond you continue as if the event as happen. Useful for simulation it is may not be it is not synthesisable . But it is useful very useful in test bench simulation and things like that.

(Refer Slide Time: 43:28)



Asynchronous Reset

41



```
process (clk, reset)
begin
  if (reset = '1') then
    q <= '0';
  elsif (clk'event and clk = '1') then
    q <= d;
  end if;
end process;
```

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q <= d;
  end if;
end process;
```

- Concurrent statements

```
q <= '0' when (reset = '1') else
     d when (clk'event and clk = '1');
```

Kuruvilla Varghese

So, let us come to this part then we have looked at the code for a flip-flop without considering the reset. And the code was like this clock begin if clock tic given and clock is equal to 1, q gets the end if, end process okay. Now we know that you know practically we need to reset**,** that is 1 point in design you should have a reset for flip-flop okay.

We will maybe briefly touch upon it, this part. Because if you do not reset properly there is no guarantee that first up all we do not know whether the q will be 1 or 0. And also there is something sometimes this painting can happen, it can gets stuck in between and all that. So, you need a good reset and you know the asynchronous reset. Asynchronous means that when you say synchronous it means that it is synchronous with the clock.

Asynchronous means any time you have said this any times you make it 1. This goes to 0, irrespective of the clock maybe high or low or the active edge come. There are some timing requirement with respective to this clock will. But let us ignore that part okay. In real life there is some timing restriction with respective. The clock we will we have not learned the basics regarding that.

So, we will ignore it for a while, so if you assert this the q become 0. And if it is like, so it has a priority over the clock okay. Anytime this is asserted then the q becomes 0. If it is 0 then it operates like a normal flip-flop. If clock come d gets q. So, let us think how to incorporate that in this code a reasonable behaviour. So, naturally that means that no more the process, we are looking for both simulator and synthesis tool together.

And , so basically the reset is asynchronous. So, reset has to be in the sensitive day list. Because any time reset happens. Then the q has to go to 0. So, we need to write the reset okay. Assume that we call it RST or something like that. We need to write RST here. Now it has priority over this business so, that code has to come before that okay. So, we what we do is that we write reset here.

And we are going to say if reset is 1, then q gets 0, else if clock tick event clock is equal to 1, then q gets d, end if okay. So, that is what is the right code asynchronous reset, process, clock and **re** clock, reset, begin, if reset is 1, then q gets 0. And if there are this is a single bit if you have a register. Parallel register you say q get others 0. Everything is made 0, else if clock tic event, clock is equal to 1.

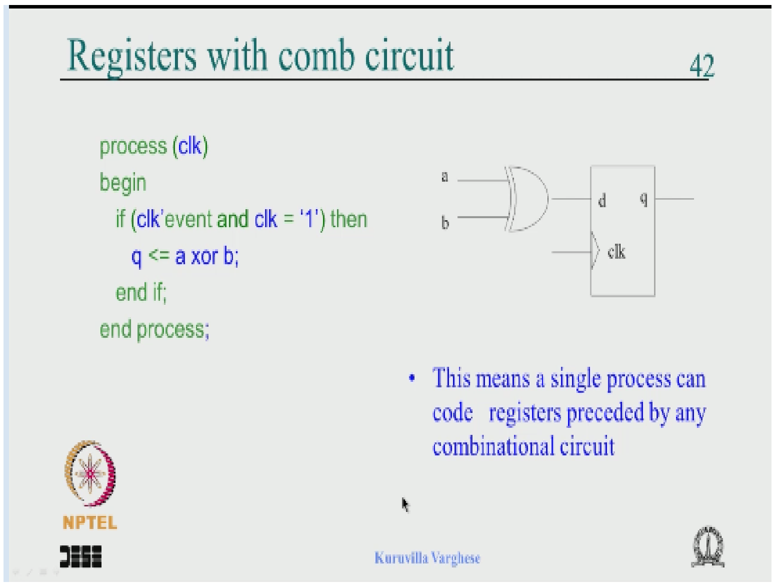
Then q gets the end if, mind you cannot say else. Because then everything is completed, we do not have kind of implied latch. So, that is why else if is specified because we want avoid the else then only that memory part is kind of specified, so there is no else, else if, clock tic event clock is equal to 1, end if. So, that rightly kind of model the behaviour of asynchronous reset.

So, that is it, that is a code and that is most useful code for a flip-flop or a register. There is no difference with parallel register you just make the q s kind of wide with whatever you require. Then you gets a and the d is equal to that. Then instead of 0 you write others 0, then you get the code for the registers. Which is with asynchronous reset.

So, can we have the concurrent statement for flip-flop with asynchronous reset. If you think yes we can say, we can use when-else which is equivalent to the if then. We say q gets 0, when reset is 1, else q get the if else q get the clock event, clock is equals to 1. So, you can say that you know q gets 0 when reset is 1, else q get d when clock event, clock is equal to 1.

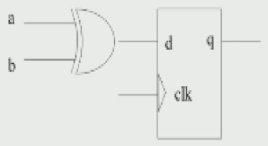
You can terminate without saying further else. So, that should give you a flip-flop with asynchronous reset you can try this with you know with the synthesis tool we are using. I will definitely show a demo of the tool.

(Refer Slide Time: 48:59)



Registers with comb circuit

```
process (clk)
begin
  if (clk'event and clk = '1') then
    q <= a xor b;
  end if;
end process;
```



- This means a single process can code registers preceded by any combinational circuit

NPTEL

Kuruvilla Varghese

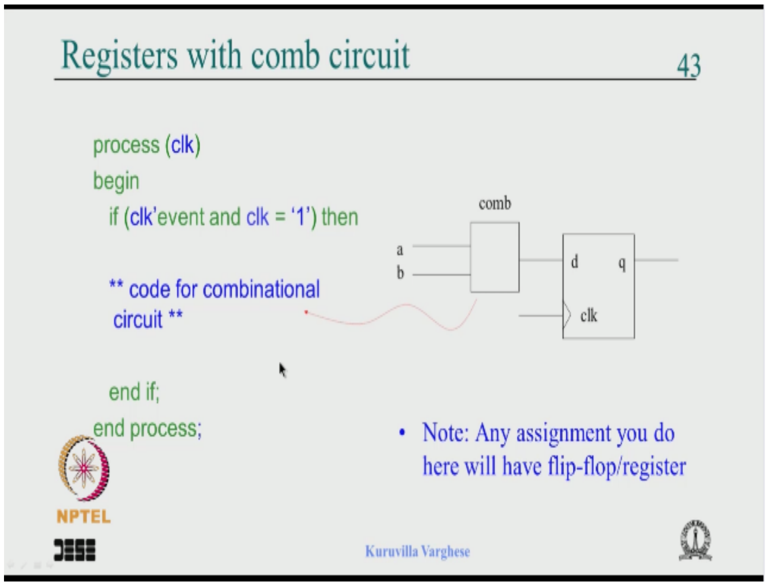
Now let us look at please look at this code first we will talk about behaviour of it. So, this say that process clock I am not showing the reset, you can add reset, it makes no difference. But which a begin, if a clock tic event and clock is equal to 1. Then q gets a xor b okay. Now this says that when the clock comes q will get some input you know. We normally write q gets d.

Which say that q gets a xor b okay. It is a perfect code, it will simulate, it will synthesise synthesis tool will generate a circuit for it. So, we know that what happens when the clock comes, when the clock comes d gets to q. So, if this has to happen what we are saying is that when the clock gets q gets a xor b means definitely **.** If that has to happen we know that the clock comes d is a one which get done for it q.

So, naturally there has to be an exclusive OR gate here with input a and b, then only this behaviour will happen, and that is what you are going to get. If you write a code like this. You will get an exclusive OR gate at the d input of the flip-flop okay. So, when the clock comes the q gets a xor b. That is what is the meaning here what it says is that you could write a single process for a flip-flop with some logic proceeding it okay.

So, you could write 1 process and underneath clock tic event and clock is equals to 1. You write q gets some combinational circuit, that comes before the flip-flop. And as I said it need to be here it is a single bit, it can be a raw registers with a much more complex circuit and you can code it using a single process. So, that is a meaning of it a single process can called registers, proceeded by any combinational circuit okay.

(Refer Slide Time: 51:27)



Registers with comb circuit

43

```
process (clk)
begin
  if (clk'event and clk = '1') then

  ** code for combinational
   circuit **

  end if;
end process;
```

- Note: Any assignment you do here will have flip-flop/register

Kuruvilla Varghese

So, let us generalise it let us say you write a process I am not showing reset you can add reset it, just to make you know if I add reset. Then I have the problem of the statement going below my screen, that is why I avoid the reset and you can add it. So, here process clock, if clock tic event and clock is equal to 1, that is a end if. Now you write any code for combinational circuit does not matter you know.

If not need not be very simple statement like that you know. You can write say a some select signal is then you say when 00 00, then you say if you write anything you want. As for as synthesis tool can make sense out of it. You write a very complex behaviour, then what happens is that, that happens like this. You get a flip-flop or a set up registers and depending on what you write .

And then you get whatever the combinational circuit you have written the code for that comes before okay. So, it is very convenient way of coding a register and a combinational circuit proceeding it okay not something after it. So, you can always combine some combinational circuit followed with the register. So it is very convenient way of writing the RTL code a register transfer logic code **.**

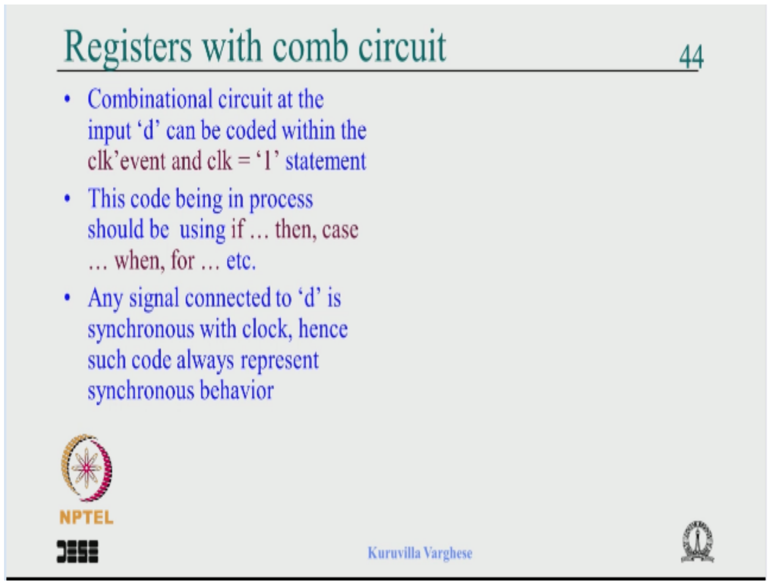
I have avoided the (()) (52:51) that is what it is you know that is nothing but the RTL code that is what I have been describing all alone even in the digital design as I emphasising that. So, here one danger you should understand that, anything you write, anything you assign. Suppose you write z and an assignment operator, z will become a flip-flop or a register.

So, any assignment you make under within this body, you will get a flip-flop free. So, very be very careful you know. Some people happily write code without thinking about it. And that will have lot of side effect okay. So, if you do not realise any assignment you make at that output of that assignment will be flip-flop. And that could be unintended okay.

Maybe you are trying to put say there is a sequential circuit and from the present state you are decoding something. And if you write the decoding part within this clock tic clock is equal to 1. You will get a set of registers or a flip-flops at the output of it. And you simulate that you will find that for whatever reason you may not note the real you may not verify the what actual circuit you have got out of it.

You going to simulation you will find that the decoder circuit comes with the delay. And then you do some adjustment to counter the delay okay. And you can end up in real soup with that. And that is dangerous, so you should now thoroughly understand anything you write here, you will get a flip-flop. That is also good, because you want to put some circuit, complex circuit here before set of registers, just write if clock tic event clock is equals to 1. Then you write the code for it, that you will get it any assignment will have a this one kind of flip-flops.

(Refer Slide Time: 54:57)



Registers with comb circuit

44

- Combinational circuit at the input 'd' can be coded within the clk'event and clk = '1' statement
- This code being in process should be using if … then, case … when, for … etc.
- Any signal connected to 'd' is synchronous with clock, hence such code always represent synchronous behavior

NPTEL
Kuruvilla Varghese

And you could write that code using any construct it does not matter. This is sequential statement so, you can use if then case when for so, use any of the construct and happily write the code that naturally comes here. So, that is what is like I have told today basically we have looked at the code for we have talked about the flip-flops for synthesis and simulation. And there is a issue with clock tic event.

Because it can represent the unintended event so, you could use rising edge in falling edge you can have equivalent concurrent statement for flip flops and for the latch as for as simulator is concern. We need d in the sensitivity less so, this works as a code for latch for both for synthesis and simulation. You could have equivalent concurrent statement like this and you could use instead of sensitivity less wait on but not both.

And these are the wait syntax is wait on wait handle wait for wait for is at definitely for simulation the combination you have to think whether it makes sense in synthesis. We have seen how to incorporate as synchronous reset basically we write put reset in the sensitivity less, before the clock event you write if then you get asynchronous reset. You can have concurrent statement equivalent concurrent statement.

And when you write something a combinational behaviour here, you will get a combination circuit before it and that shows how to write register proceeding the combination circuit. You write the code like that. So, please go through this part thoroughly understand it grasp the concept well. Thank you I wish you all the best.