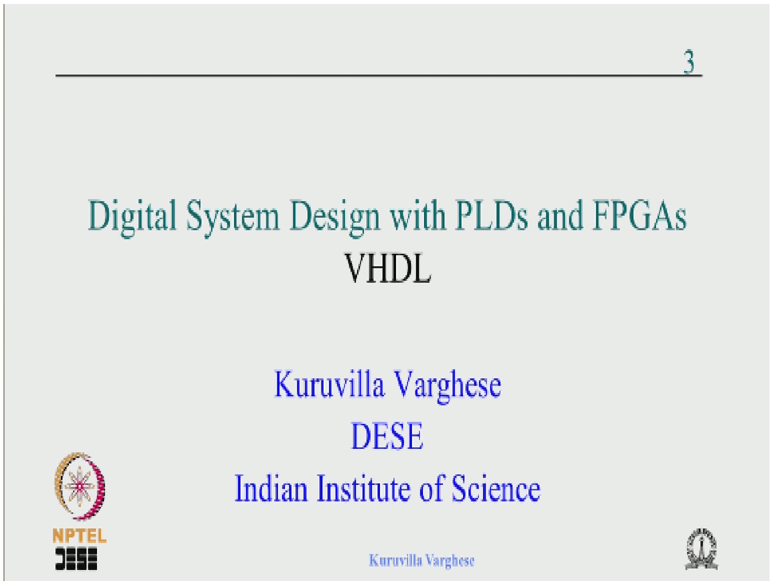
Digital Systems Design with PLDs and FPGAs
Kuruvilla Varghese
Department of Electronic Systems Engineering
Indian Institute of Science - Bangalore

Lecture-21
VHDL Examples

So, welcome to this lecture on VHDL.

(Refer Slide Time: 00:24)



3

Digital System Design with PLDs and FPGAs

VHDL

Kuruvilla Varghese

DESE

Indian Institute of Science

NPTEL

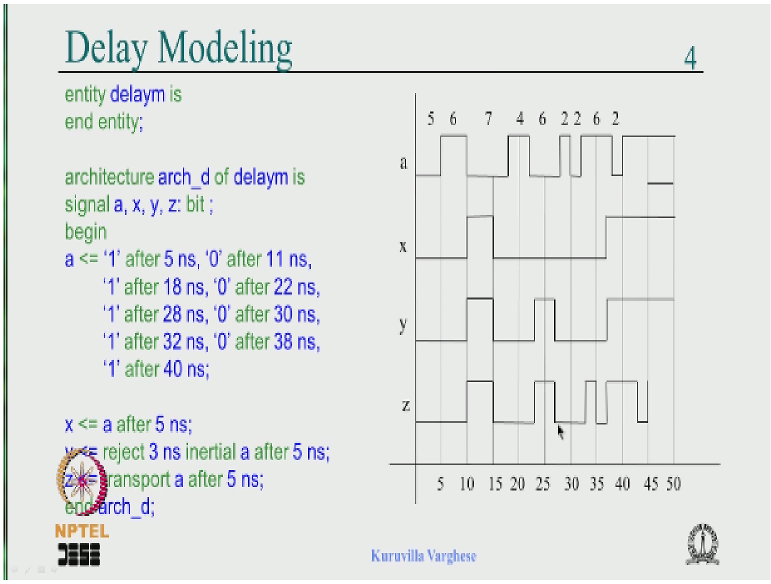
Kuruvilla Varghese

In the course digital system designed with PLDs and FPGAs last class we have looked at the we have completed the delay modelling and we have started some examples. So, that we apply what we have learned and we could not complete that part. So, I am planning to compete it, not only designing something. But you know give a code and kind of work out what is the possible circuit which implements or given to a synthesis tool what it synthesise.

So, that you get it experience you know back and forth given a circuit how to code it for synthesis or given a code how to kind of make out what is implemented. It is very useful when you really work in a design team, you may have to work with something which is designed earlier. And most of the time it is done like that you know you have. You are in a team you join a team there is something going on.

And it will be given some responsibility many a times; you will end up with some very log of VHDL code. And you have to add to it, modify it, debug it and rarely there will be enough documentation. So, to know looking at the code what is underline hardware the represent is the very useful thing to do that for we are going to do today. In addition to even designing some circuit. So, let us quickly run through what we have done last class and so, that we can continue kind of smoothly from there. And let us move on.

(Refer Slide Time: 02:20)



Delay Modeling

4

```
entity delaym is
end entity;

architecture arch_d of delaym is
signal a, x, y, z: bit ;
begin
a <= '1' after 5 ns, '0' after 11 ns,
      '1' after 18 ns, '0' after 22 ns,
      '1' after 28 ns, '0' after 30 ns,
      '1' after 32 ns, '0' after 38 ns,
      '1' after 40 ns;

x <= a after 5 ns;
y <= reject 3 ns inertial a after 5 ns;
z <= transport a after 5 ns;
end arch_d;
```

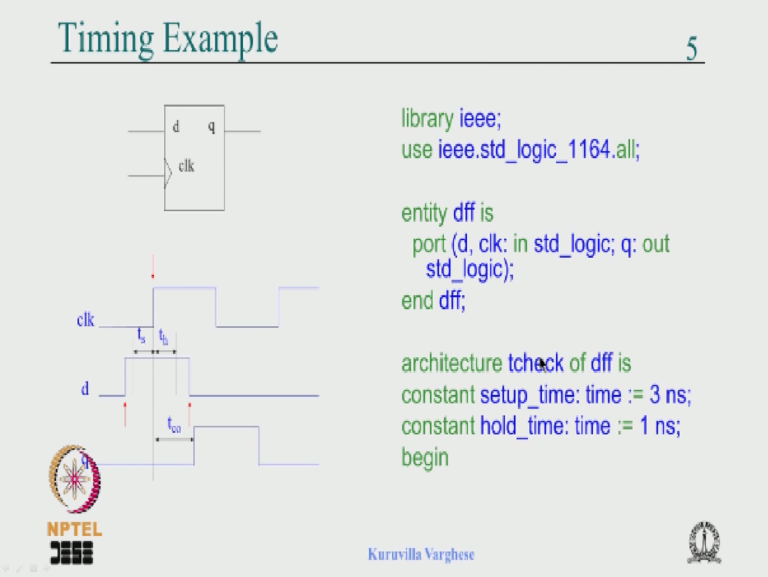
NPTEL

Kuruvilla Varghese

So, this is what we have seen kind of an example program to illustrate the various delay model. So, we have created a signal a, it is a kind of empty entity. Because we just wanted to see the waveform and we have using instead of code we are using everything a signal. We form an signal a with some pulses. And that is what is shown here and x is a after 5 nanosecond.

So, it has a pulse width, minimum pulse width 5 nanosecond propagation delay 5 nanosecond. And so, you apply you get a delayed version of this with all the pulses less than 5 removed. Next is reject 3 nanosecond inertial a after 5 nanosecond wherein is exactly saying. But everything below 3 is kind of rejected, so you get this. And this is the transport delay with the 5 nanosecond delay. So, you get z as a delayed version of a that is what is shown here.

(Refer Slide Time: 03:32)



Timing Example

5

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

entity dff is
  port (d, clk: in std_logic; q: out
    std_logic);
end dff;

architecture tcheck of dff is
constant setup_time: time := 3 ns;
constant hold_time: time := 1 ns;
begin
```

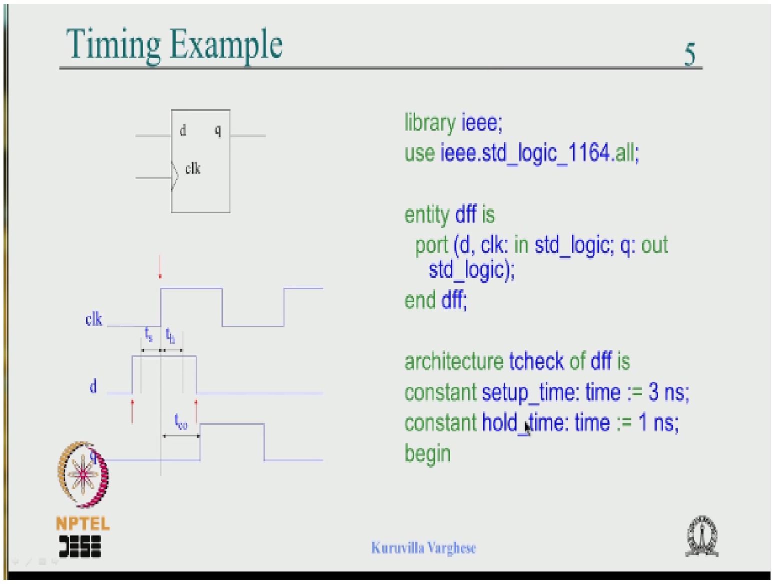
Kuruvilla Varghese

And we also have looked at how to write a VHDL code to not only to synthesise, but to verify some timing condition on it is input okay quite useful. Because sometime you may have to like you have some timing violation happening in the simulation. And suppose it is random and you want a catch it, and you can do like that to this way though the timing analysis will show up but definitely this is another way.

Static timing analysis can report the whole time violation the slack and all that, but yet this is a practice and useful thing we are doing. So, our aim is to write a code which checks the setup and hold time violation of the d with respect to the clock. So, basic idea is to keep track of the event, this particular positive edge of the clock and any event on the d**.** The idea is that when there is an positive edge of the clock subtract the last event on d make sure that it is greater than setup time.

Similarly whenever d changes subtract the last event on the clock then you know make sure that is greater than the whole. So, we have the entity here, we define a 2 constants, setup time and hold time to be conveniently kept in one place. So, that you can modify it one shot.

(Refer Slide Time: 15:12)



Timing Example

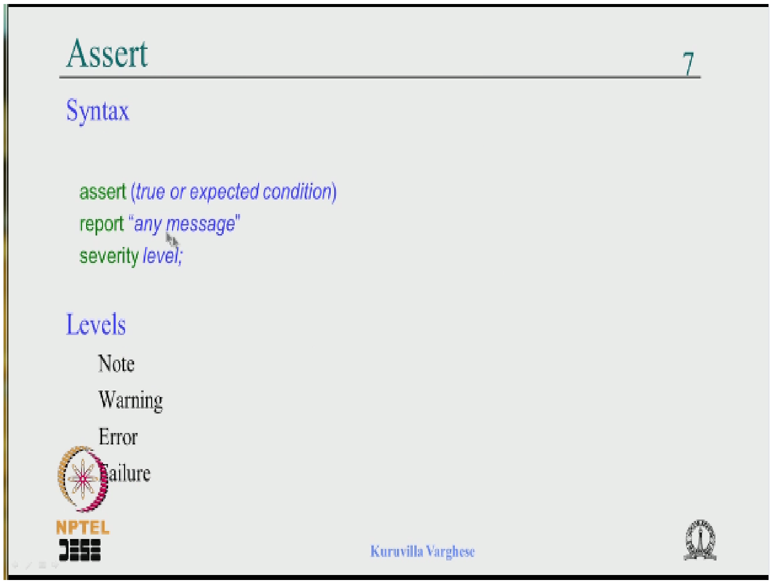
```
library ieee;
use ieee.std_logic_1164.all;

entity dff is
  port (d, clk: in std_logic; q: out
    std_logic);
end dff;

architecture tcheck of dff is
constant setup_time: time := 3 ns;
constant hold_time: time := 1 ns;
begin
```

Kuruvilla Varghese

And we have a process with both d and the clock in sensitivity less which is required because we have to catch the event on d and this is the behavioural code and this is the main code which checks for the whole time violation. If there is an event on d make sure assert is asserting the true condition that the time of the event – the clock event is greater than the whole time and update the time and if there is a positive clock edge assert.

That the time between that time and the last event on d is greater than or equal to setup time and update the clock time and that is how it is done and as I said at the 0 nanosecond we do not want it report an error. So, we make sure that, that is avoided by this statement.

(Refer Slide Time: 06:07)



Assert

Syntax

```
assert (true or expected condition)
report "any message"
severity level;
```

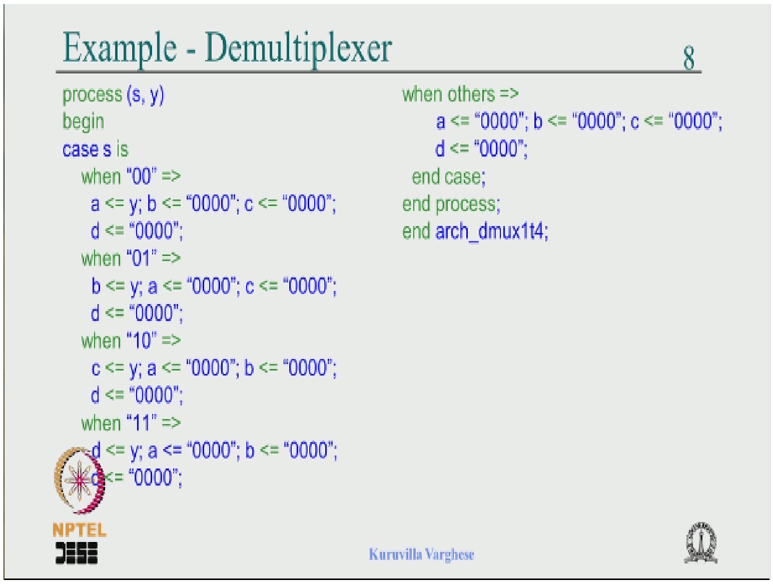
Levels

Note
Warning
Error
Failure

Kuruvilla Varghese

And we have seen the syntax of assert, a true or expected condition and then id it is violated for report a message and you can specify what should the tool do by this level for not an warning it is just printed on the console. And for a error and failure it just terminates the simulation process. So, that is the syntax of assert.

(Refer Slide Time: 06:36)



Example - Demultiplexer

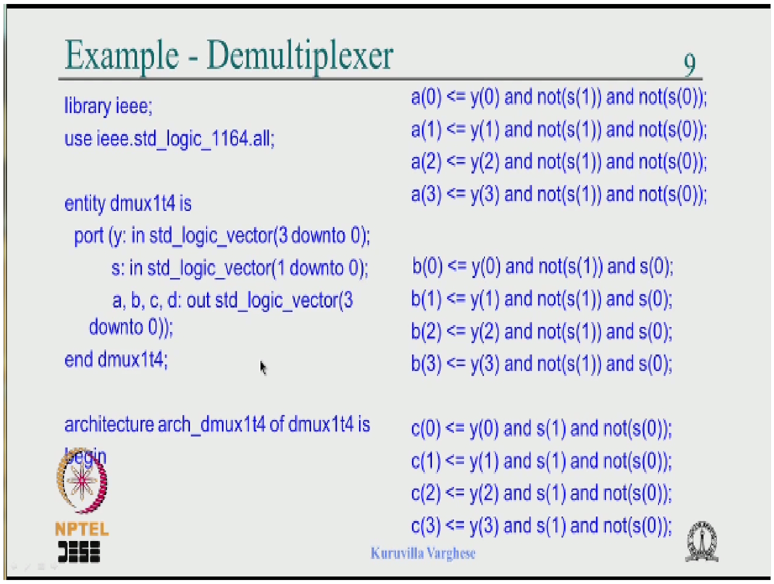
8

```
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;
```

Kuruvilla Varghese

And then we started with examples we have seen the a VHDL code for a de-multiplexer. The most proper code is this when you have s is a select line, y is the input a, b, c, d are the output. When it is 00, y goes to a rest all are 0. 01 d goes y goes to b, then so on you know c, d. So, we have this is the ideal kind of coding for the most proper coding for the de-multiplexer because there no priority. And for all the select line values we have specifying the output and all the outputs are functioned of the same s and y. So, it is nice to combine this way, but there are many other ways to do it.

(Refer Slide Time: 07:21)



Example - Demultiplexer

9

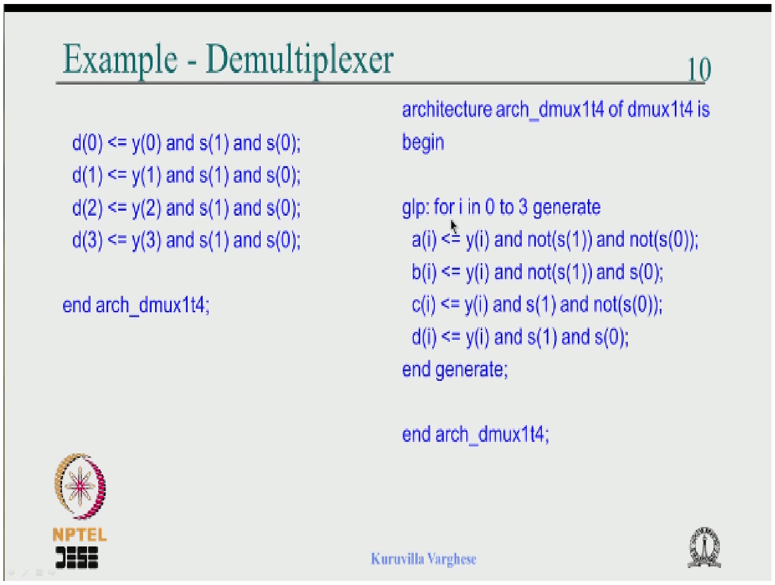
```
library ieee;
use ieee.std_logic_1164.all;

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
```

```
a(0) <= y(0) and not(s(1)) and not(s(0));
a(1) <= y(1) and not(s(1)) and not(s(0));
a(2) <= y(2) and not(s(1)) and not(s(0));
a(3) <= y(3) and not(s(1)) and not(s(0));

b(0) <= y(0) and not(s(1)) and s(0);
b(1) <= y(1) and not(s(1)) and s(0);
b(2) <= y(2) and not(s(1)) and s(0);
b(3) <= y(3) and not(s(1)) and s(0);

c(0) <= y(0) and s(1) and not(s(0));
c(1) <= y(1) and s(1) and not(s(0));
c(2) <= y(2) and s(1) and not(s(0));
c(3) <= y(3) and s(1) and not(s(0));
```

Kuruvilla Varghese

And we have seen you can write equation for a, b, c and d in terms of the input and the select line. So, this is a0 is yo not of s1 and not s0. And when it comes to d it will be d0 gets y0 and s1 and s0. Because it is 11 and since it is a 1 to 4, 4 bit de-multiplexer you have to write the equation for all the 4 bits. And now looking at this equation you know seeing this symmetry you can make out that this can be easily put it in a generate loop with index going from 0 to 3 for this and this is fixed anyway similarly for this and so on.

(Refer Slide Time: 08:13)



Example - Demultiplexer

10

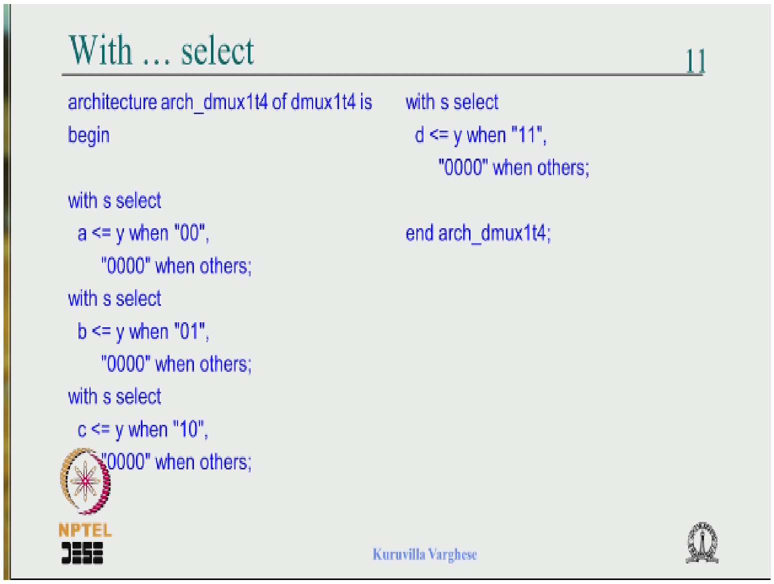
```
 d(0) <= y(0) and s(1) and s(0);
 d(1) <= y(1) and s(1) and s(0);
 d(2) <= y(2) and s(1) and s(0);
 d(3) <= y(3) and s(1) and s(0);

end arch_dmux1t4;
```

```
architecture arch_dmux1t4 of dmux1t4 is
begin

glp: for i in 0 to 3 generate
  a(i) <= y(i) and not(s(1)) and not(s(0));
  b(i) <= y(i) and not(s(1)) and s(0);
  c(i) <= y(i) and s(1) and not(s(0));
  d(i) <= y(i) and s(1) and s(0);
end generate;

end arch_dmux1t4;
```

Kuruvilla Varghese

So, we can combine everything into generate loop like for i in 0to 3 generate, a of i is y of i and that means you just take the first statement put i here, i here. Similarly take this i here, i here and so on. Then you get this and please remember that this is not a loop in a conventional stands that like in a sequential language you go you know kind of execute 1 and execute the other it just a concise way of writing this like this you know. This is literally replicated 4 times and you get all the big equation. So, it is a short hand.

(Refer Slide Time: 08:59)



With … select

11

```
architecture arch_dmux1t4 of dmux1t4 is
begin

with s select
  a <= y when "00",
      "0000" when others;
with s select
  b <= y when "01",
      "0000" when others;
with s select
  c <= y when "10",
      "0000" when others;
```

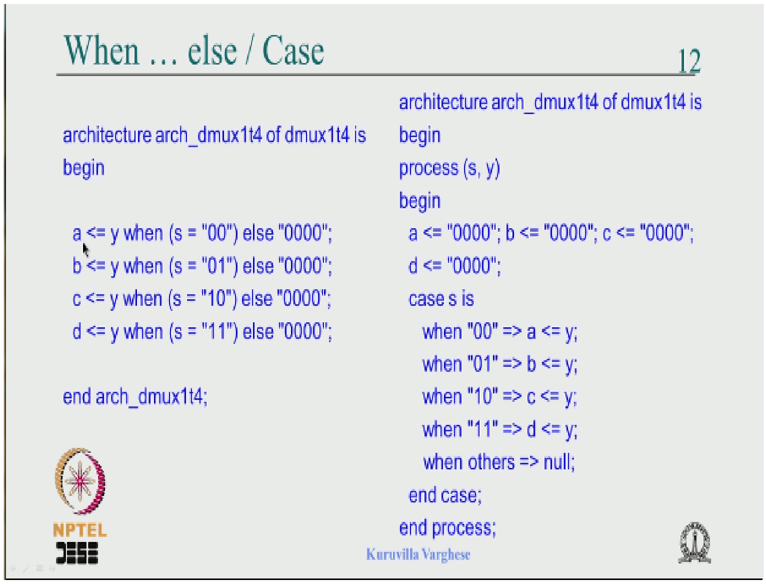
```
with s select
  d <= y when "11",
      "0000" when others;

end arch_dmux1t4;
```

NPTEL

Kuruvilla Varghese

You have to write a with select for each output a, b, c and d. This is quite right, because no problem because even in the hardware a output can be separated from the circuit for the a can be separated from the b. Absolutely no issue you write like that you get the you get the circuit you want. But, the synthesis tool has to infer now they saying that abcd is part of a single high level block whole de-multiplexer. And that is only trouble with this code for these simple things the tools will do. But, you have to be careful similarly you have the when else.

(Refer Slide Time: 09:46)



When … else / Case

12

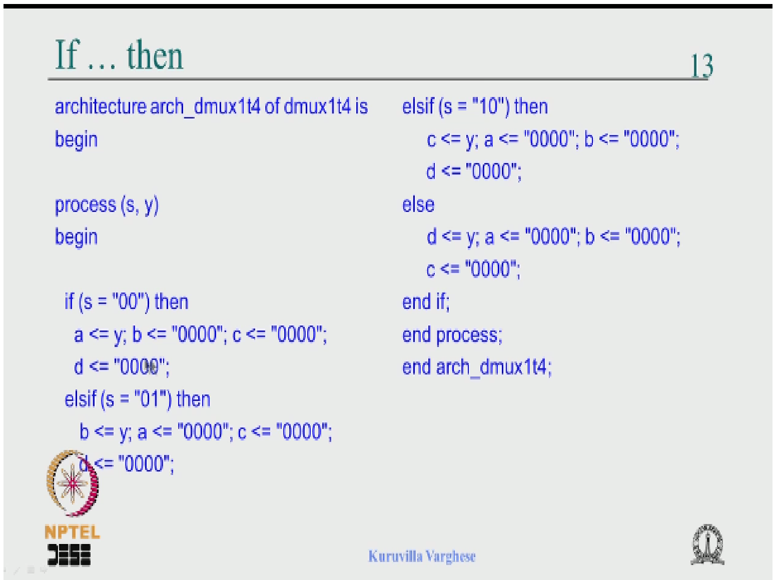
```
architecture arch_dmux1t4 of dmux1t4 is
begin

  a <= y when (s = "00") else "0000";
  b <= y when (s = "01") else "0000";
  c <= y when (s = "10") else "0000";
  d <= y when (s = "11") else "0000";

end arch_dmux1t4;
```

```
architecture arch_dmux1t4 of dmux1t4 is
begin
process (s, y)
begin
  a <= "0000"; b <= "0000"; c <= "0000";
  d <= "0000";
  case s is
    when "00" => a <= y;
    when "01" => b <= y;
    when "10" => c <= y;
    when "11" => d <= y;
    when others => null;
  end case;
end process;
```

Kuruvilla Varghese

You have 4 of them a gets y, when s is 00, else 00. Because we only talk about a output. Similarly b, c and d another version of the case we do not say 00 everywhere. We make it at the beginning then only change is indicated here.

(Refer Slide Time: 10:08)



If … then

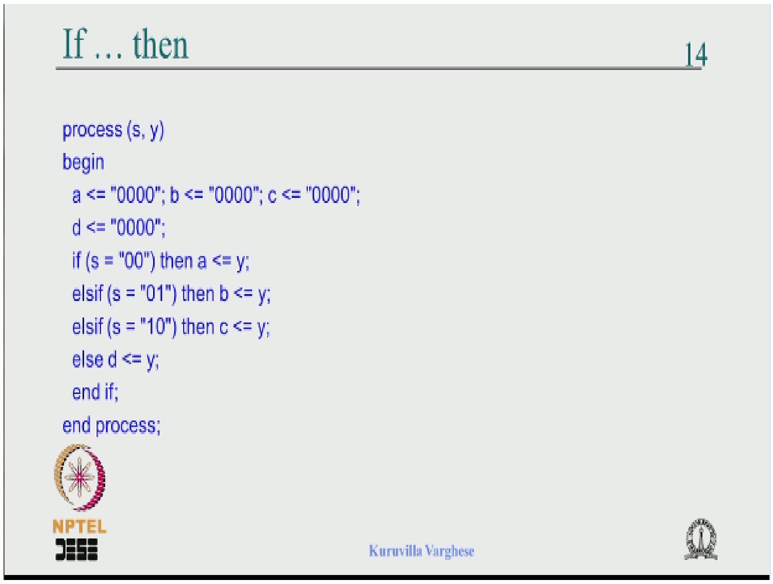
13

```
architecture arch_dmux1t4 of dmux1t4 is
begin

process (s, y)
begin

 if (s = "00") then
   a <= y; b <= "0000"; c <= "0000";
   d <= "0000";
 elsif (s = "01") then
   b <= y; a <= "0000"; c <= "0000";
   d <= "0000";
elsif (s = "10") then
   c <= y; a <= "0000"; b <= "0000";
   d <= "0000";
else
   d <= y; a <= "0000"; b <= "0000";
   c <= "0000";
end if;
end process;
end arch_dmux1t4;
```

Kuruvilla Varghese

Then instead of case you can use if there is like you know that b for as a priority definitely as it is return you say s is 00 then, s is 01 that means not of. But since we are a enumerating all possible values of s and when you say and you know s is not of 0 and 01. Then everything reduces to same as case. So, it is not a problem the syntax does not give a problem. But still the most elegant code will be to use the case.

(Refer Slide Time: 10:46)



If … then

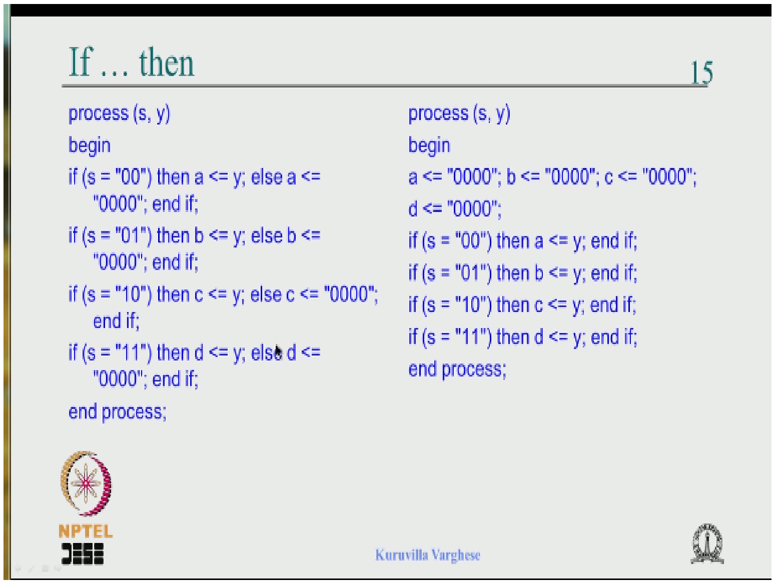
14

```
process (s, y)
begin
  a <= "0000"; b <= "0000"; c <= "0000";
  d <= "0000";
  if (s = "00") then a <= y;
  elsif (s = "01") then b <= y;
  elsif (s = "10") then c <= y;
  else d <= y;
  end if;
end process;
```

Kuruvilla Varghese

And as in case you can initialise all the values at beginning and you know represent only the change here**.**

(Refer Slide Time: 10:56)



If … then

15

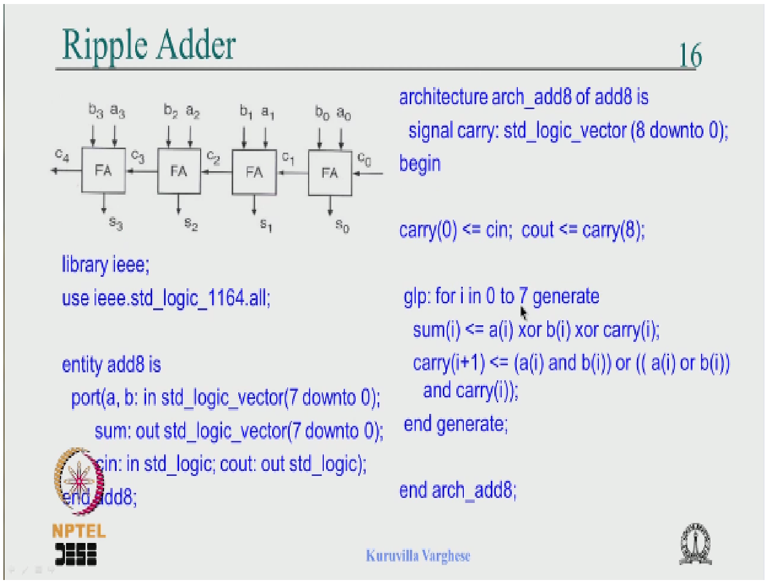
```
process (s, y)
begin
if (s = "00") then a <= y; else a <=
    "0000"; end if;
if (s = "01") then b <= y; else b <=
    "0000"; end if;
if (s = "10") then c <= y; else c <= "0000";
    end if;
if (s = "11") then d <= y; else d <=
    "0000"; end if;
end process;
```

```
process (s, y)
begin
a <= "0000"; b <= "0000"; c <= "0000";
d <= "0000";
if (s = "00") then a <= y; end if;
if (s = "01") then b <= y; end if;
if (s = "10") then c <= y; end if;
if (s = "11") then d <= y; end if;
end process;
```

Kuruvilla Varghese

You could use another way is using any for each of the output separate like you say. If s is 00, then a gets y else a gets 00 end if. It is exactly like writing the with select or when else. Once again it is technically correct but the ideal code would be to combine everything. All the outputs together as a function of a and y another version is you know everything is initialise at the beginning and only the change is indicated here.

(Refer Slide Time: 11:34)



Ripple Adder

16

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

entity add8 is
  port(a, b: in std_logic_vector(7 downto 0);
     sum: out std_logic_vector(7 downto 0);
     cin: in std_logic; cout: out std_logic);
end add8;
```

```
architecture arch_add8 of add8 is
  signal carry: std_logic_vector (8 downto 0);
begin

carry(0) <= cin;  cout <= carry(8);

 glp: 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;

end arch_add8;
```

NPTEL

Kuruvilla Varghese

And this is what we have maybe stop at the last lecture. We have we are writing an 8 bit ripple adder okay for the figure only shows a 4 bit ripple adder to save space but the idea is same you know you have at the cast leads at full adders. Full adder states ab inputs, a carry input and it produce a sum and a carry out. And a carry out goes to the next carry in of the next full adder and so on okay.

And the issue with this ripple adder is that if you take the c4 or sum3 it uses c3 and when you say you take the case of sum3 in the worst case it propagates from c0 to c1, c1 to c2, c2 to c3 and c3 to s3. So, there are if you assume 2 levels of circuit in each of and or. Then you have 8 gate delays by the time you come here. And you can imagine if you can go to the s7 you will have the 16 gate delay.

And that adds a lot of delay okay, so there are circuits ripple adders which sorry the adders which gives less delay and so this is simple you have the sum equation, sum of i is ai xor bi xor carry i, carry i + 1 is ai and bi or ai or bi and carry i which when you expand you get ab, ac and bc cb in the carry and to make the code neat . We have a cin which is c0 and cout which is say in our case ca.

So, to make the code clean not right everything in a single loop and not to write anything outside we can define a carry which is 9 bit. We assigned carry 0 as cin and the carry out as sorry c out as a carry 8. Otherwise we will end up writing a you know 1 assignment 1 reputation for with cin for the kind of some 0 and the carry 1, then the last one with the carry you know cout and all that.

That is avoided by this game. Now also I have been mentioned it is very unusual like a all the (()) (14:21) may times when you read the textbook you will design lot of combinational circuit which is all designed with the truth table, when it comes to the adder somehow surprisingly you kind of quickly make a full adder then cascaded. But that is I mean if you go by the principles of what you have learned.

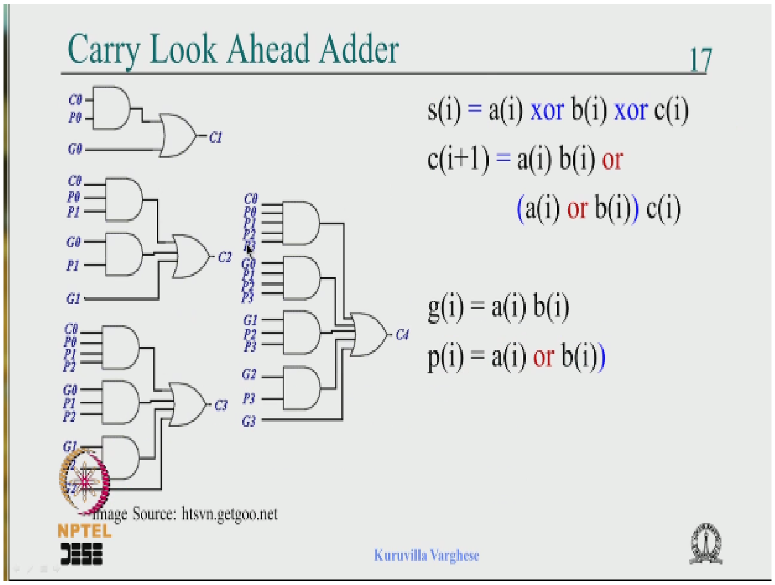
If you want an 8bit adder then you make an truth table for you know a and b which are 8 bit. So, it will be 16 bit, 64 k truth table yeah 11 definitely we cannot work it on the paper . Then you implement a circuit which essentially should give you an AND, OR circuit which is 2 level or 4 level or whatever depending on all the complexity. So, if you do that what you get is a carry looked adder.

And since that is complex one should you know go for a modular approach at least in the case of the adders it is possible. Because of this structure you add something then you have a carry which is going to the next level. And the next level you do the same thing. So, because of the modularity because of the symmetry it is possible to design a kind of modular ripple adder, modular adder.

It may not be possible in every case, things can be not very symmetrical, symmetric or regular. So, many times it is not amendable to such key. But this is at least this amendable to that, so we do this okay. So, that is a basic game. Now as I said now as your drop the kind of carry look ahead adder is derived from here not an issue.

But the basic game is that in a carry look ahead adder is that when it comes to this stage. Instead of using something already defined this stage will work with b1a1, b0a0c0 to generate both s1 and c2, that is a basic game okay or when you come to the stage third stage instead of just using b2, a2, s2 and already form c2 we will bill we will use b1, a1, b0, a0, c0, c1 everything I mean to generate. So, that is one shot it is generated not rippling through the stages that is a game **.**

(Refer Slide Time: 17:04)



Carry Look Ahead Adder 17

s(i) = a(i) xor b(i) xor c(i)

c(i+1) = a(i) b(i) or

(a(i) or b(i)) c(i)

g(i) = a(i) b(i)

p(i) = a(i) or b(i))

Image Source: htsvn.getgoo.net

Kuruvilla Varghese

So, let us quickly maybe that gives a kind of revision and some practice in coding that is what you know shown here. So, this is the adder equation s of i is a of i xor b of ix or c of i, no confusion when b of ripple adder or carry look ahead adder the thing is changed the game is not then nothing complex with complicated with that. That the problem is here, so we tagged that.

So, c of i + 1 is aibi or ai or bi and ci and this is now for clarity this is called generate g aibi, because if both are 11 you know that the sum is 0 and there is a carry out definitely there is a carry out okay **.** Irrespective of the carry input to that stage is 0 or 1 to both are 11, the sum is 0, carry out is 1. If the carry input is 0 no change. If the carry input is 1 sum will become 1 carry out is 1.

So, absolutely no change of that is called generate. The carry is generated at this stage, but if any one of them is 1, ai or bi is 1. Then the carry is generated if input carry is 1 okay. So, c1 is 1, if a0 or b0 either of them are 1. And c0 is 1 you know that is a game. So, we call that is propagate that means if ai or bi the carry before you know the carry input is propagated to the carry output. So, we define 2 thing gi which is aibi, ai or the bi.

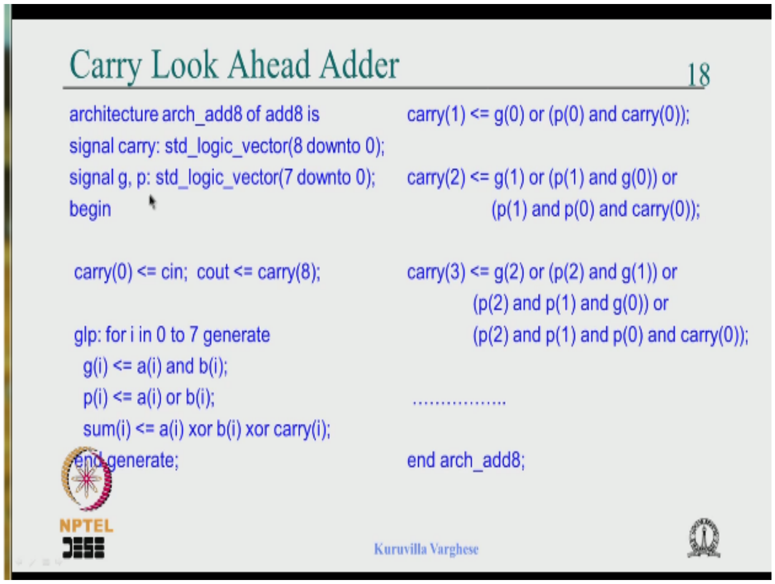
Now the game is very simple like a you say you take this c2, this is nothing but a1b1 or a1 or b1 and c1 you know usually the c1 comes from the previous stage okay. So, what is done is that you substitute the equation for c1 here and expand it and you get a 2 level AND OR okay. That is only the game you know that, that is what is the carry look ahead adder. So, the problem is that as you go the higher indexes.

Because it is you know expanding more and more product terms will in the implementation that is what is shown here, you can work it out I think maybe you have studied. So, it is very simple c1 is nothing but g0 which is a0bo or kind of a0 or b0 and c0, so that is p0 okay. Similarly when you come to the next stage c2 is a1b1 that is g1 okay or a1 or b1 and g0, g0 is nothing but what is generated in the previous which is the carry input to this stage.

So, which is nothing but a0bo okay or a1 or b1 and a0 or b0 and co and that is how the propagated from the beginning okay. So, when it comes to the third stage you have kind of 3 AND gates and 1g2 which is nothing but the kind of a3 sorry a2b2 and so on. So, it goes like that, so even if you put say g0 is nothing but an AND gate here, p0 is an OR gate here. So, you get a 3 stages of delay for any of the stages, there is the rippling is avoided.

But it is a very complex circuit and now at least at a simplistic level we will not be able to write a loop, because the sum is no issue, sum is simple. Bu the carry is something which is which go and expands. So, with the signal as we learn now it is very difficult to write a carry look ahead adder you can write with variables. Since we are not looked at it we will delay that kind of for a moment . So, we will just write look at this code you know the equivalent code in VHDL that is what we are going to do.

(Refer Slide Time: 21:44)



Carry Look Ahead Adder

18

```
architecture arch_add8 of add8 is
signal carry: std_logic_vector(8 downto 0);
signal g, p: std_logic_vector(7 downto 0);
begin

 carry(0) <= cin;  cout <= carry(8);

 glp: for i in 0 to 7 generate
   g(i) <= a(i) and b(i);
   p(i) <= a(i) or b(i);
   sum(i) <= a(i) xor b(i) xor carry(i);
 end generate;
```

```
carry(1) <= g(0) or (p(0) and carry(0));

carry(2) <= g(1) or (p(1) and g(0)) or
            (p(1) and p(0) and carry(0));

carry(3) <= g(2) or (p(2) and g(1)) or
          (p(2) and p(1) and g(0)) or
          (p(2) and p(1) and p(0) and carry(0));

 ................

end arch_add8;
```

Kuruvilla Varghese

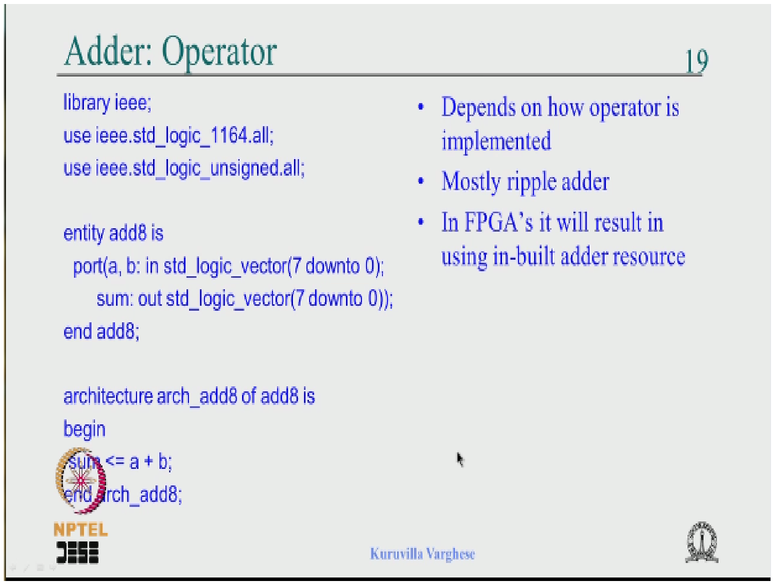
So, not showing the entity, entity is same you have like in the previous ab is 8 bit, sum is 8 bit, there is a cin and cout there is a signal which is carry internal, carry internal signal which is kind of ninth bit, so that all holes moved here. But in addition we need now g and p generate and propagate which is for each stage. So, we have an 8 bit vector as early we in the earlier case we have the carry 0 which is cin, carry out cout which is carry 8.

Now we write a loop not only for the sum, but also gi and pi, because that is regular. Every stage needs that, only thing is that maybe the second stage use the not only g1 the g0 also it is used. But does not matter we can generate it parallely in a loop. So, this is the loop index loop label for i in 02 generate g of ai and bi, p of i is ai or bi and sum of i is ai xor bi xor carry i. Now we are force to write it each carry equation separately.

The carry 1 is g0 or p0 and carry 0 that is shown here g0 or p0 and carry 0. Similarly carry 2 is g1 or p1 and g0 or p1and p0 and carry 0, g1 or p1 and g0, p1 p0 or c0. Similarly it goes and you have to write up to carry 7 actually matter, because many a times you write an 8 bit or 16 bit or 32 bit maybe even 64 bit luckily we are not yet in 128 bit we are the number representation as on now like double precision.

It can be 64 bit, so it not a big deal even if you do not write a loop you can still implement that just because the code is longer, you will not end up with any longest I mean any biggest circuit luckily this is hardware not a c code. But definitely if you write this scheme in a c it might kind of similar style in c you might get a very you might take longer time to kind of execute such a code, not a problem in hardware though we will definitely see how to write that in the loop.

(Refer Slide Time: 24:43)



Adder: Operator

19

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

entity add8 is
  port(a, b: in std_logic_vector(7 downto 0);
      sum: out std_logic_vector(7 downto 0));
end add8;

architecture arch_add8 of add8 is
begin
sum <= a + b;
end arch_add8;
```

- Depends on how operator is implemented
- Mostly ripple adder
- In FPGA's it will result in using in-built adder resource

Kuruvilla Varghese

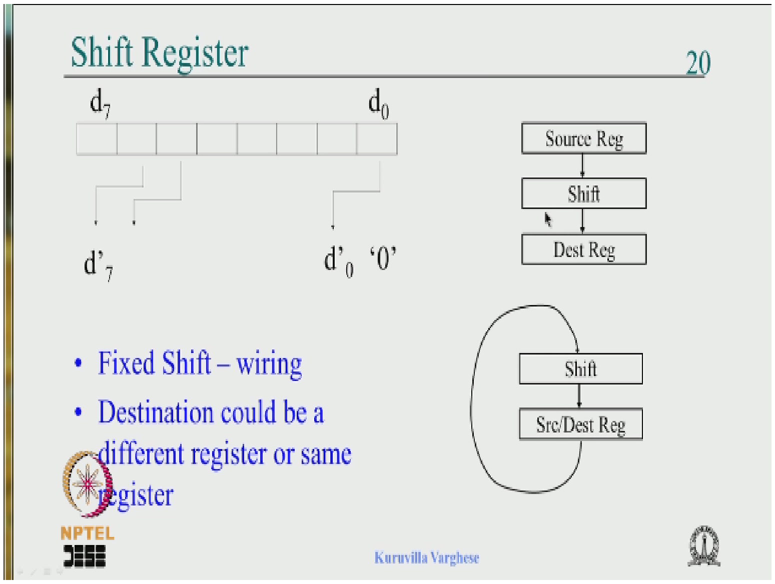
So, but after all you know this is for a practice you know you will have looked at the ripple adder code we have looked at the ripple adder, carry look ahead adder and but when it comes to the VHDL normally you want an adder you just write you know it is an 8 bit adder then ab is 8 bit vector sum is 8 bit vector just say sum gets a + b, if you want a carry out we have discussed in the arithmetic section how to write that you know.

We have seen that you can adopt that scheme. But now when you write such a scheme it depends in the library code of the tool or this + is written okay. It will mostly will end up with a ripple adder when it comes to FPGAs of PLDs at least in FPGA there are kind of dedicated adder resource. We will see that they will use that built in resource have to implement the ripple adder which will be much faster than implementing a any other adder.

Where b it a carry look ahead or carry select or carry skipper whatever you say a simple +, in the case of FPGAs will be much faster. Because it is in silicon built into the silicon. But if do anything else, if you use the programmable wires which is going to add delays and it is going to occupy lot of logic resources which need to be interconnected and it is going to be delay.

So, ultimately when it comes though we have done some practice with the adder, when it comes to an adder really we are going to use most of the time the + operator within that.

(Refer Slide Time: 26:38)



Shift Register

20

- Fixed Shift – wiring
- Destination could be a different register or same register

Kuruvilla Varghese

So, let us take an example of a shift register those who have done an undergraduate program sometime it will little bit you are confuse about the shift register. Because somehow this shift is kind of couple register you know you have learned a block called shift register in a very particular very restricted fashion. And it very simple thing is may very complicated okay. So, when you say shift okay.

Suppose you have a register, 8 bit register d7, d6 up to d0 okay, this is the MSP, this is the LSP. Suppose you want a left shift by 1 okay you like this is stored in a register you want to output say suppose this is kind of save for a it is there it is not changing and you want a shifted version of this left shift by 1 okay, it is very simple you take the d6 as the shifted version of for the 7, d5 is used as d6 or dash 6, d0 is used as sorry.

This is I am sorry this is d dash 1 I will change that, so this is d dash 1 and you upend a 0. So, basically when you say shift alone, it is just a wiring when you say fixed shift okay, when you say shift left by 1 or shift right by 2. It is always a wiring there is no hardware is applied okay, absolutely take 6 or 7 here 5 as 6 and 0 as 1 okay shift a then you upend a 0. So, it is just of wiring.

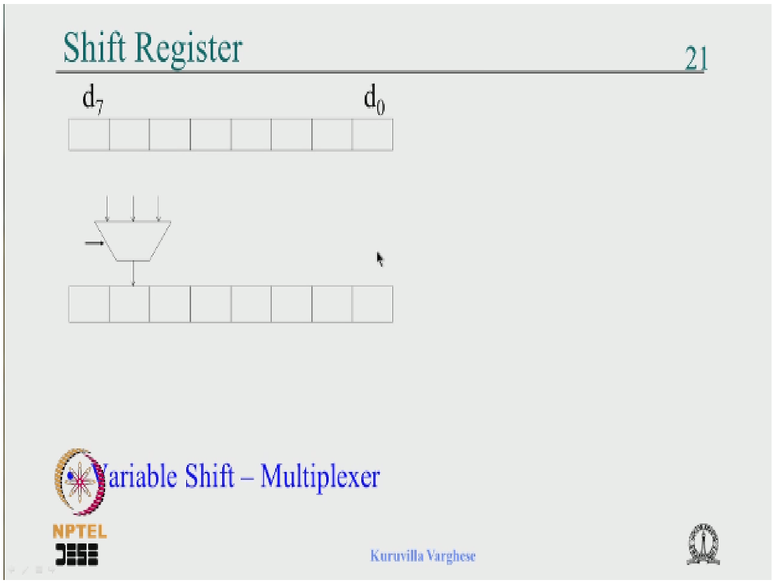
But now you want to store and suppose this is changing and this is changing continuously screening as in a pipeline then you need to store this somewhere. And that can be stored in a separate register mostly it is done like that rarely that it is put back to the same register in a series circuit you will be any computation there will be stored in a separate register. But it can be push back to the register and that is what you have studied in undergraduate.

So, anytime you say shift this particular picture comes to your mind and sometime it will create little kind of confusion. So, that is what I have shown here, you have a source register some kind of wiring for a fifth shift. And destination register and wiring could be you know whichever way want maybe is everything is shifted by 4 that can be kind of you know all the LSP lease significant go to MSP when it comes here maybe this part can come down here depending on the requirement okay.

So, there are it depends on the application, the computation. But this is what you have learned where the source and the destination register is same. That means you take the qs you have register which is kind of okay this maybe can call q or being the output. But I hope the concept is clear, then here you take the q shift it and put it to the d of the same register okay that is a shift register.

Now suppose want happens if it is the shift is variable that means when some control signal is active then you say let us call one control signal called left or right. If the left or right is 1, then do a shift left by 1, if it is you know if it is 0, then shift right by1. So, it is a the shift is variable now, either shift this way or shift that way. That definitely you cannot wire it. Because now depending on the control signal you have to get this or that. That shows a mugs okay you have to have a mugs.

(Refer Slide Time: 30:55)



Shift Register

21

Variable Shift – Multiplexer

NPTEL

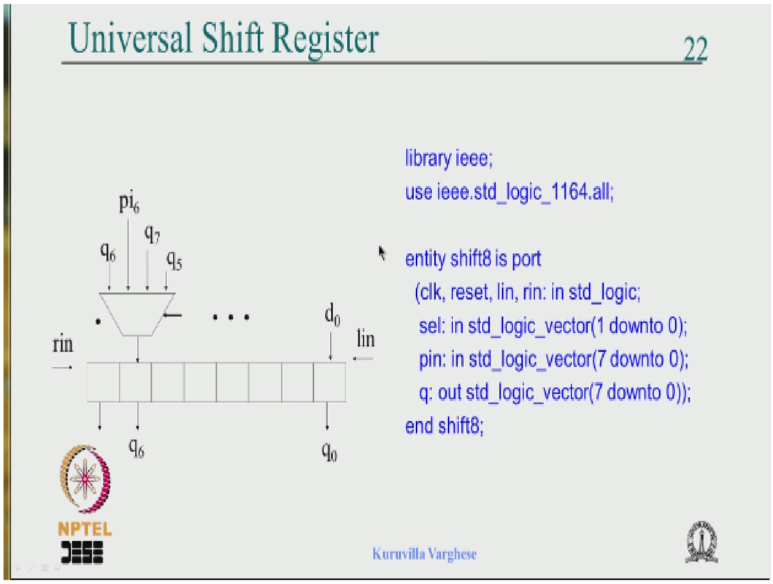
Kuruvilla Varghese

So, when you have a variable shift, then you need to use a multiplexer. So, I am showing a scheme wherein either say you take the q6 the d6 part I am showing just 1, then with a shift left then the q5 is push here if it is shift right, the q7 is pushed if not if either of that it is not true. Then maybe the same thing can be push okay. Now it depends on the select line, if select line is only 1 bit, then you can choose between this 2 if it is 2 bit.

Then if it is for 00 you retain the same you pass a same it is 01 do the shift left, so 10 you do the shift right 11 again maybe the pulse is same or whatever okay. So, when you have a variable shift you use multiplexer in this case it is an 8 bit register. So, you can use 8 multiplexers which is all 3 to 1 okay. But it is possible to view a instead of 8 kind of single bit multiplexer maybe you can view it as a kind of huge a to 1 8 bit multiplexer.

That means that you assume a huge 8 bit multiplexer here, the output is going to all the 8 registers. And there is here to all the inputs and so on okay. You can view it that way also does not matter and you can have the coding also written that way whichever is convenient.

(Refer Slide Time: 32:39)



Universal Shift Register

22

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

entity shift8 is port
  (clk, reset, lin, rin: in std_logic;
   sel: in std_logic_vector(1 downto 0);
   pin: in std_logic_vector(7 downto 0);
   q: out std_logic_vector(7 downto 0));
end shift8;
```

Kuruvilla Varghese

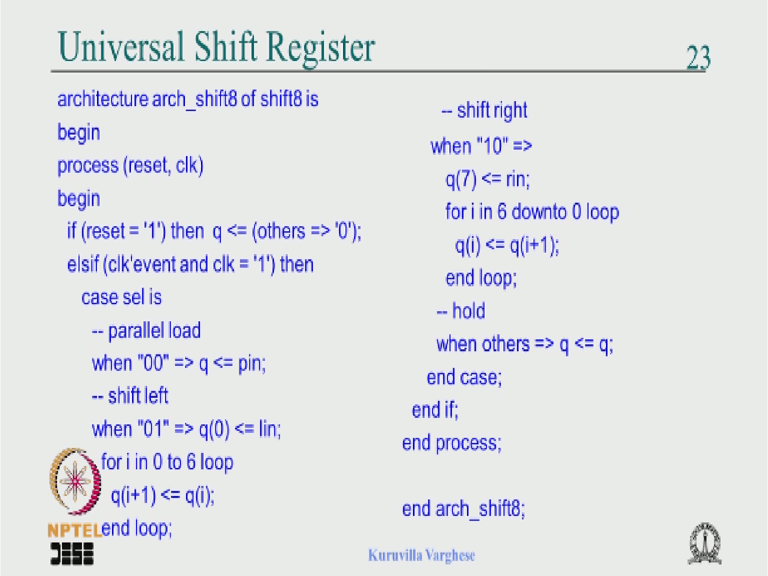
So, let us look at a universal shift register which allows you to do the shift left, shift right and you can hold the value with the same or you can parallely input some value okay like you can load it parallely you can shift left, you can shift right. So, we have a register which goes from d7, d0 when you are shifting left there is an input called l in which goes to q0, q0 go to q1 and so on okay. And similarly when you are right shifting the rin goes to the q7 and q7 go to q6 and so on.

And okay I am not shown the clock, but the clock is there, reset is there. So, the select line, also we have a take the case of a 6 or 7 stage q6. Then the d6 you have a mugs here which is 4 to 1 mugs. So, we have 2 select lines when it is say some value q5 for shift left comes, when you want a right shift the q7 is put in and if you want parallely load something then this input is coming from the outside pi6 is loaded the sixth bit.

And if it is not you retain the value you re-circulate the value okay. And this mugs is repeated kind of 8 times. So, that is what the real the hardware is and now we write the code and when we write the code you could write it as you know one kind of 4 to 1 mugs single bit repeated 8 times or as I said you can have a you can think of a huge mugs where the 8 bit output is there and input is also 8 bit.

Because after all that we are connecting all the inputs down here. So, but it is you can expand it to bring clarity to it. So, whichever way you view it, but that is hardware. So, this is the library close for a and this is the entity where clock and reset are the input, left in right in is a input all single bit, select line is because the 4 you know the 4 to 1 multiplexer you have 2 bit, parallel in is 8 bit. Because there are it is an 8 bit register and the q is also 8 bit okay. Now you are you can sense already that okay.

(Refer Slide Time: 35:29)



Universal Shift Register

23

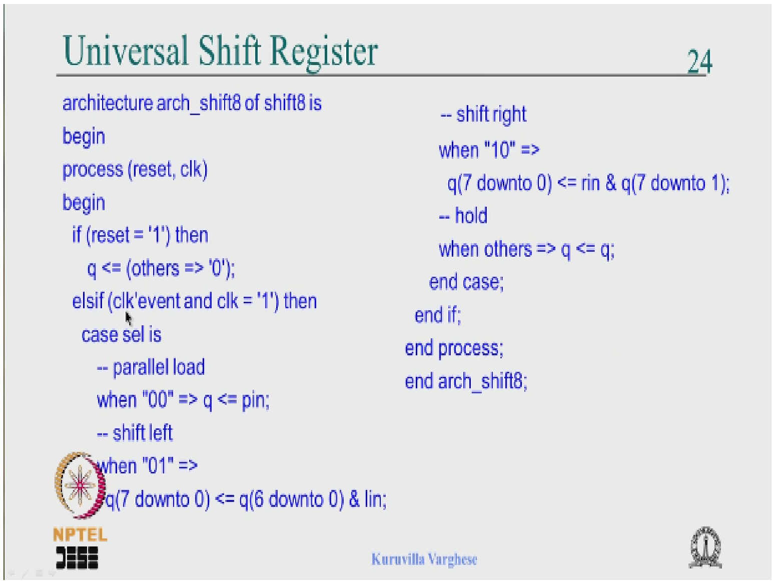
```
architecture arch_shift8 of shift8 is
begin
process (reset, clk)
begin
 if (reset = '1') then  q <= (others => '0');
 elsif (clk'event and clk = '1') then
    case sel is
      -- parallel load
      when "00" => q <= pin;
      -- shift left
      when "01" => q(0) <= lin;
       for i in 0 to 6 loop
         q(i+1) <= q(i);
       end loop;
       -- shift right
      when "10" =>
        q(7) <= rin;
        for i in 6 downto 0 loop
          q(i) <= q(i+1);
        end loop;
      -- hold
       when others => q <= q;
     end case;
   end if;
end process;

end arch_shift8;
```

Kuruvilla Varghese

Now I am showing there could be a little yes okay. So, we have to maybe kind of you know assign a signal and we have to declare a signal called q here which have not done. But we can do that instead a port here, we will declare a signal you imagine that this as some other name and we have a signal q here.

Because we have somewhere we are going to say you know the q is come on the right hand side not the right thing to do. So, we have to declare this a signal sorry for the minor error but the assume that the q is signal here, so now we say process, reset clock begin if reset is 1, then q is others 0.

(Refer Slide Time: 36:15)



Universal Shift Register

24

```
architecture arch_shift8 of shift8 is
begin
process (reset, clk)
begin
 if (reset = '1') then
   q <= (others => '0');
 elsif (clk'event and clk = '1') then
  case sel is
   -- parallel load
   when "00" => q <= pin;
   -- shift left
   when "01" =>
    q(7 downto 0) <= q(6 downto 0) & lin;
   -- shift right
   when "10" =>
    q(7 downto 0) <= rin & q(7 downto 1);
   -- hold
   when others => q <= q;
  end case;
 end if;
end process;
end arch_shift8;
```

We are resetting all the register contents else if clock tic event clock is equal to 1, the n case select is parallel load when 00 we have parallely loading q gets p in. That means q 7 down to 0 get pi p in 7 down to 0 that is a meaning it. Then that is a parallel load shift left when 01, q0 get a l in. So, that is the q0 q0 is getting the l in and then we can write a loop where for i in 0 to 6 loop qi + 1 get qi.

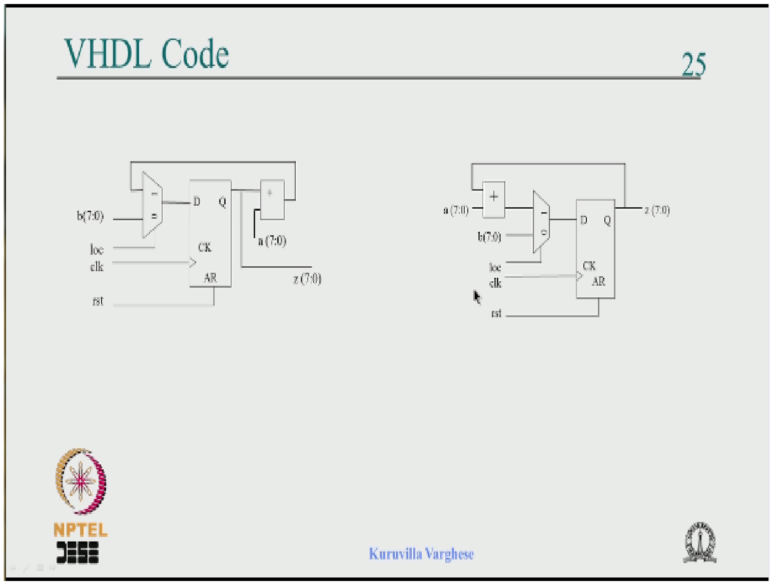
For the shift right q7 get the right in you can say right in and q6 get the q7 that means qi gets qi +1 into the opposite to that end loop and if others. In other case that means when 11 is there are all other case sq gets q okay. Since we have saying q is coming at the right hand side it has to be a signal it cannot ne a an output. So, for simple cases you do not verify. So, that is why it happens.

So, you just write the code now you can also write you know that is a code and this how you should design you know simple thing. But when you start you draw the blog diagram and you write the code and after while you know this will be in you r mind for simple things at least . So, you can still you know code it with this is in keeping this detail in the mind.

But the complex thing at least all the block should be shown. So, that everything is clear. So, let us write this instead of the loop as a single mugs you know that is what we have going to write you know that is how the code is here. If reset is 1 then q gets others 0, else if clock tic event clock is equal to 1 case select is when 00 q gets p in. But when 01we are going to see it as a single mugs now q7 down to 0 get q6 down to 0 and lin okay that is what is shown here.

Q7 down to 0 gets q6 down to q0 and lin now this part gets this part you know that is what is shown here. And for the right shift it is opposite q7 down to 0 get rin which comes at MSP and q7 down to 1 okay. So, rin go to q7, q7 go to q6 and so on. When others q gets q okay, that is a code and this is the most elegant code as for as the universal shift register is concerned.

(Refer Slide Time: 39:04)



VHDL Code

25

NPTEL

Kuruvilla Varghese

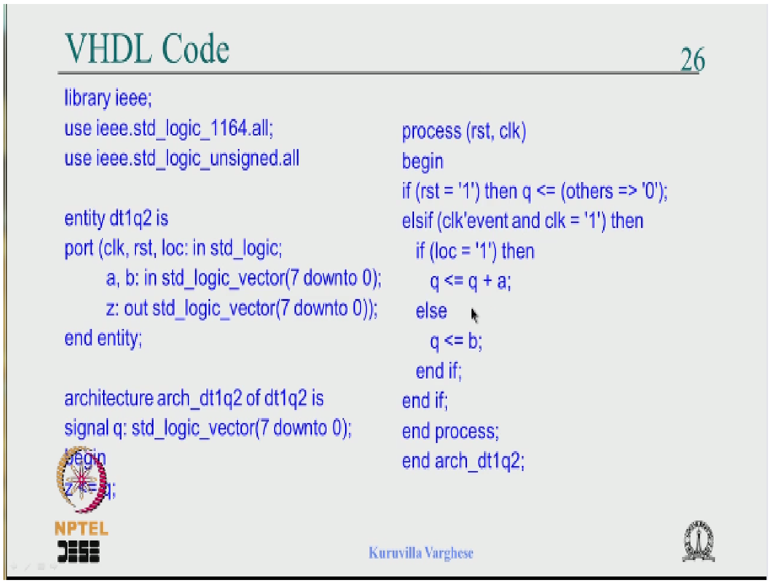
So, let us take another example okay which is little bit kind of arbitrary, so please have a look at the circuit okay. Now this shows first of all an 8 bit register. Because you know that this output is said 7 down to 0. So, this definitely are 8 flip-flops, 8 de flip-flops, the clocks are connected together reset is connected together 8 bit register, the output is combined added with an a 7 down to 0.

That goes here to a 2 to 1 mugs, and when a signal called LOC is there then when LOC is 0 the b gets loaded LOC is 1 this some get loaded. Now we have seen that a set of registers with a preceding combinational circuit can be coded in a single process. So, we are going to write a single process for this. But the question is what about this okay. Now this comes looks like at least in the picture it looks like it comes afterwards.

But then you know that this is a kind of circle and this is also behind though it use as a input called q this + this adder comes here you know. So, you can transform this blocks schematic like this you know it is same we have you know the + is put here and z is taken here you know it is exactly same. Now we can look how to code it. So, we know that how to code a flip-flop you write a process with clock and reset.

If reset is 1 then this z will be others 0 or the appropriate you know we might use a signal because this is used internally. So, we can define some output assigned that and all that. Then we are everything is synchronous it is upon the clock. So, we say in the clock event clock is equal to 1. If LOC is 1 sorry LOC is 1 then we say this output gets a+z okay else it gets b that is all okay. So, I suppose you get a picture, so we say that if LOC the output is upon the clock, it is under the clock a+z else it is b. So, it is very simple code.

(Refer Slide Time: 41:58)



VHDL Code 26

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

entity dt1q2 is
port (clk, rst, loc: in std_logic;
      a, b: in std_logic_vector(7 downto 0);
      z: out std_logic_vector(7 downto 0));
end entity;

architecture arch_dt1q2 of dt1q2 is
signal q: std_logic_vector(7 downto 0);
begin
z <= q;
```

```
process (rst, clk)
begin
if (rst = '1') then q <= (others => '0');
elsif (clk'event and clk = '1') then
  if (loc = '1') then
    q <= q + a;
  else
    q <= b;
  end if;
end if;
end process;
end arch_dt1q2;
```

Kuruvilla Varghese

So, we have the library declaration the package 1164 and unsigned mainly because we are going to use this + and + is overloaded for standard logic in this particular library. And this is a entity okay. Now mind you I have put some kind of arbitrary name for it because when you kind of give a example like that you should not write some obvious name.

So, that the students can detect it. so, this how I write the kind of questions when whenever I give question I do not write a counter or legible name or even you know these name sometime I change. So, that is not easily obvious like when you go back from a VHDL code to the blog diagram. So, here the ports are clock and reset and LOC are single bit input.

Ab which is 8 bit input, z is an you know 8 bit output as I said since we are going to use z as an input we are going to define a signal called q which is also 8 bit and we assigned z the q to z **.** so, that we can use q being a signal internally. So, this is the main a the code a single process for this. So, if clock event process reset clock begin. If reset is 1 then q is others 0, else if clock event clock is equal to 1.

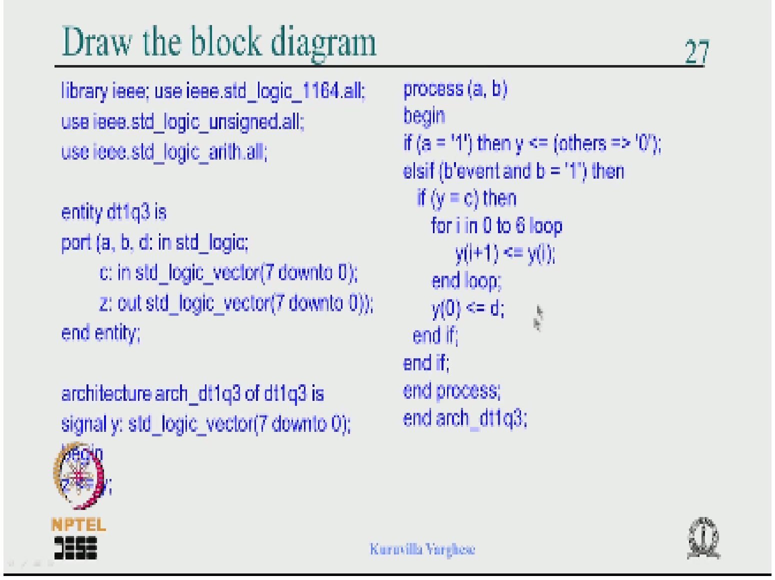
Then we say if lock is q gets a+z else b. So, that is it if lock is 1 then q gets q+ a yes. This is instead of z we have q +a. Because q is a signal q+a else q gets b end if. So, this is the one which is representing the 2 to 1 mugs and this + represent the adder which comes before the 2 to 1 mugs. And everything being in the clock event clock is equal to 1 it goes to a register or you gets goes to the d of the registers all the bits you know.

The q7 goes to the d7 and so on okay. And say end if for this one, this is a end if for this end process for this end the architecture okay. As I said you intend it properly, so that people can make out after an experience you know you look at the code you know you can draw the blog diagram. So, that is how this particular blog diagram is easily very easily. So, just you have the picture here you can write the code or you can even tell your team members to write a code.

Suppose many a times you have designing you do not have to write the code you know you come to a block level RTL diagram and give it to somebody, you can code it and the coding is not a big deal , designing is a big deal. Once you get to know come to this level then the coding is 121 you know just look at the picture and you write few lines of code it does not matter you now you see this much you have a register 2 to 1 mux and Plus and he does something that is all coded in a very few lines 1, 2, 3 maybe finish line you know three four five lines it can be coded.

Because this very few lines required for 2 to bring the hardware, to code the hardware okay, now let us do the opposite okay hear what you have done that you are given the circuit on the secured we have go ahead adder VHDL code and we have done that for and very simple loan secured like the ripple adder carry look ahead adder the shift register and now I have given some kind of random circuit without worrying its functionality and we have coded it symbol.

(Refer Slide Time: 46:06)



Draw the block diagram

27

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

entity dt1q3 is
port (a, b, d: in std_logic;
      c: in std_logic_vector(7 downto 0);
      z: out std_logic_vector(7 downto 0));
end entity;

architecture arch_dt1q3 of dt1q3 is
signal y: std_logic_vector(7 downto 0);
begin
z <= y;
```

```
process (a, b)
begin
if (a = '1') then y <= (others => '0');
elsif (b'event and b = '1') then
  if (y = c) then
    for i in 0 to 6 loop
      y(i+1) <= y(i);
    end loop;
    y(0) <= d;
  end if;
end if;
end process;
end arch_dt1q3;
```

Kuruvilla Varghese

As soon as you put this kind of the block diagram then you can code it. Now we do the opposite you know somebody has given us ok so VHDL code you have to kind of make out what the circuits or the block diagram or the RTL whatever you call is ok, that is the basic game here. So let us see and as I said I have made the entity something input something, some kind of random names have not given any kind of sensible name to it.

I am not shown something is input, something is reset nothing it is just a, b, c, d. So let us see whether we can make out what is the kind of circuits. So we have the library close package 1164 for standard logic package unsigned maybe for the I do not know what may be the just thought the + if there is + and standard logic arith maybe for this kind of relational operator maybe.

I have to check you can check that and this is the entity and in that we have 3 single bit a, b, d, single bit input, 1 input which is C which is 8 bit, 1 output which is Z which is also 8 bit ok. Now in architecture we have given some name or underscore this name of this is the name and we declare a signal y which is also 8 bit. So naturally one can assume that something related to Z and the beginning that z gets y.

So y is nothing but z we are kind of circumventing the trouble with output signal that cannot come on the right hand side. So the moment the Y comes with nothing but z. Now we have a process of the main code which has a and b in the sensitivity list. So we are sure that a and b are input ok and you said begin and you say if a is 1 then y is other 0. Else if we b tick event and b=1 then and that is end if here.

Ok so if we are very clear looking at this code at this is nothing but a register and y being in the assignment is an 8 bit register where is the reset asynchronous reset and b is a clock so we can already draw a 8 registers with a connected to a synchronous, b connected to the clock, ok. Now whatever is written here comes to the d because that is synchronous, that is written within it and you see that it say that y which is the output itself equal to c which is input.

That means whatever was a register output is taken is compared with c, if it is 1 okay if that is 1 then you see yi+1 is yi and y0 is d so it is a kind of shift or a mux which is doing the shift is a single bit shift which is the left shift the least significant bit get d. So that is all, so it is a register where is a reset, b is clock at the input is shift register, the shift register work if y=z that means otherwise it remember its own value.

We say we do not say else, so it remembers its own value, so that means y=c go to a 2 to 1 mux which is the y=c output is a select line of that mux, at the input of the select line is a shift or the same value goes to the input, that is it, so that is what is shown here. We have an 8bit registers of those these are thick lines 8 bit, the output is y which is same as z, the clock is be reset is a and the y goes to compare with c and output of the select line with select among when it is 1 then the shifted version of the y along with the d, y6 down to 0 and d goes to the y.

Otherwise it will retains itself you know it recirculated itself, that is what is the code from here. So maybe know you have one more nested if then if there is another if inside that comes before ok. You know that this is the highest priority if you write one more if that comes before that I know that is symbol like that goes like that I know the you start with hear the first second third and so on, ok.

So that is how the block diagram is very simple if you nothing to worry like you if somebody kind dump a huge code to you and you do not know what to do then you sit quiet go from top to bottom at the beginning you will be thoroughly confused you will cause your boss and do not worry sit with the code and look at it again start to write you know the look at this signal start writing what are the signal, what are the sub blocks, maybe the process maybe procedures.

You start making a simple note of each block what could be that and start drawing pictures gradually things will emerge is the clarity will bring and come up and you will be able to decode all the second maybe after drawing the circuit you will even find that you could be coded much better way in a very simple way, elegant way, concise way and which is documentation wise better and so on.

So that is that is how why have illustrated this and ok let us take maybe another example I don't know whether I will be able to complete this but let us look at it, please let us start with it so here is a little more complex okay have cooked up something so that to little bit confused you but into two kind of to make some sense order ok. So we have library 1164 package and sign package.

Because somewhere we are going to class or something like that some entity you, v are the input which is 4 bit, W is output which is 8bit ok. Now I tell you if you get this done in a test paper or an exam you should apply your mind you are not the one thing in real life where you have to get you the answer quickly though I am not kind of convinced that the testing somebody is complete and he is by doing things fast you know doing things fast represent some experience may not be the b intelligence of capability.

But an experience guy can do things faster which is many times you do not have to design you know so fast as I have sprint or something like that, but it looks the present situation looks like that you know everybody is kind of working hard to keep the Moore to work every 2 months have to have a new cell phone which is not really required to frankly in my opinion, but when you have such a problem you should my advice is that you should guess probably what is this kind of circuit which takes two 4 bit value and one 8 bit value.

Sometime and looking at that will bring some clarity have to do some kind of investigation somebody called that whatever you should be old give a gas so let us come to architecture declaration region, now there is something funny happens if there is a type b type 1 is array 3 down to 0 of standard logic vector 3 down to 0 ok. Now we are declaring d type 1 as something which is a four values each of which is 4 bit ok.

Now and we have declared a signal call y which is same as this that means it essentially means y3 is a 4 bit vector, y2 is a 4 bit vector, y1 is a 4 bit vector, y0 is a 4 bit vector, ok. So for whatever reason it is defined like that that is how it is define, you know you have 4 values each of which is 4 bit and we have declared y, so you have y3 y2 y1 y0 or otherwise yo, y1 y2 y3, each of which is 4 bit.

And when it comes to the this one a another data type which is type d type 2 is something similar is array 3 down to 0, so it is a four values of all location or 7 bit. So this is four locations of 4 bit, this is 4 locations of **7** sorry 8 bit in 7 down 0 to 8 bit and we have declared a signal x which is of this type d type 2 that means you have x3 x2 x1 x0, each of which is kind of 8 bit ok. So we have declared 2 signals y which is y 3 y2 y1 y0 4 bit, each of which is 4 bit.

And x3 x2 x1 x0 each of which is 8 bit and we are declaring a constant call temp which is a 4 bit vector which is initialize to 0 that means this is nothing but like 4 bit 0, four 0 ok, that is what is declared here maybe now we have we are coming to the end of the lecture. So we will see what is the code here and we will kind of look at it how to decode what is written in the code.

Come out with us with the proper circuit for it, that is what we are going to do so, that gives a good practice that my idea going from there circuit to the code, code to circuit and so on. So in this today's lecture we have seen basically the carry look ahead adder, a simple adder with the + operator then can we have looked at the shift register which is fixed shifting which is nothing but the wiring with register.

Then the variable shift with multiplexers and then we have seen a universal shift register we have put some arbitrary block diagram and wave return the code and now we are again back to some code to work out the block diagram. So I suggest now look at these examples and work out your own example and practice it ok. So I wish you all the best and thank you.