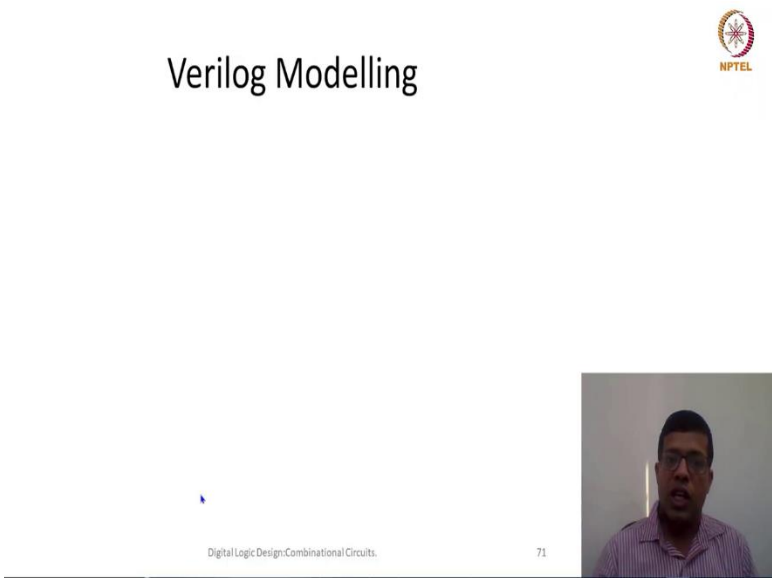
Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology Ropar
Modelling BUS in Verilog

(Refer Slide Time: 0:16)

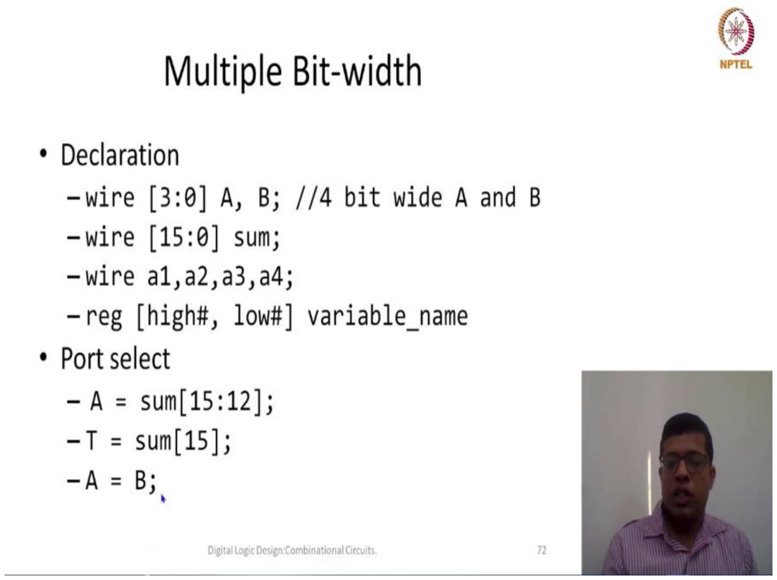


Verilog Modelling

Digital Logic Design:Combinational Circuits. 71

Now the second part of this lecture that now we need to model, we have to model all these adder circuits or adder like circuits then how do we model? So, so far in very low we understand how to instantiate gates, how to use primary gates, how to use data flow modeling, how to use structural modeling, how to connect gates from each other. Now, what additional concept rick would be required here? The most important concept which would be required here that we have to use arrays or we have to use bit vectors. So that means more than 1 bit. We need to require as a input as well as output.

(Refer Slide Time: 1:03)



Multiple Bit-width

- Declaration
  - wire [3:0] A, B; //4 bit wide A and B
  - wire [15:0] sum;
  - wire a1,a2,a3,a4;
  - reg [high#, low#] variable_name
- Port select
  - A = sum[15:12];
  - T = sum[15];
  - A = B;

Digital Logic Design:Combinational Circuits. 72

So how to handle those bit vectors so that I will quickly spend some time. Now this multiple bit width first thing is declaration. So in declaration, we can define like a regular wires and register. So here whenever we are defining wire in the square bracket, we right what is the bit width. So we write what is the high number of the bit and this is the low bit. So, whatever we are writing here will become more significant bit, whatever we are writing here it becomes 0, least significant bit.

So when we are writing like this 3 colon 0 that means it is a 4 bit wide a signal A and B, so we can, if we want to define this 4 bit we can have multiple of signals all of them would be 4 bit wide. Similarly, I want to define something which is 16 bit wide then I can say square bracket 15 colon 0. And this a1 a2 a3 a4 are the standard wire so basically when we are not writing anything in this square bracket, that is what we have been doing in our previous Verilog assignments there the default size of my variable is single bit.

So they are all single bit variables. But if we put this square bracket and put this colon that means it is a bit vector or we can call it a bus also. So, generically so instead of wire there could be (reg) also this registered this type could be interchangeably used for the declaration. So in general we are saying that this is the MSB, this is the LSB least significant bit, this is the most significant bit.

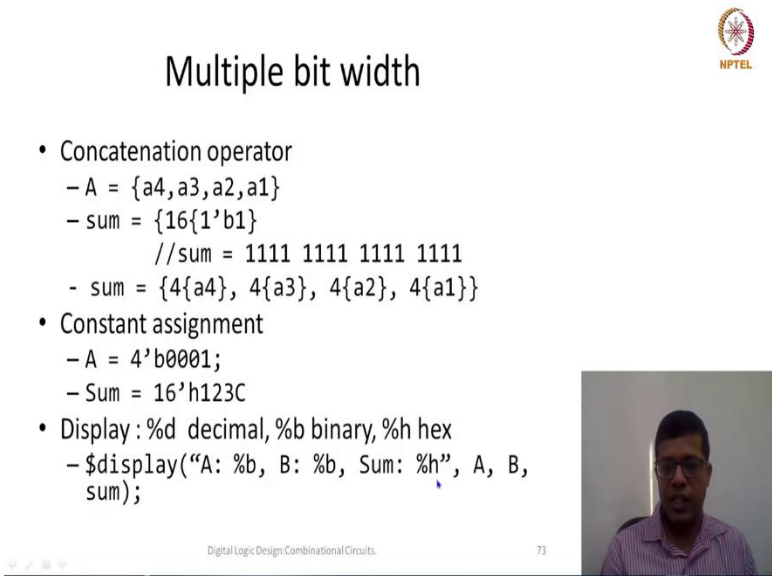
So then they could be question can I define it like 0 colon 3? Yes, you can define it. But in that case the 0 would be treated, so the 0 th bit would be the most significant bit while the whatever is written here that is the least significant bit. The pattern would still remain same. So, now if I have these multiple bits and I want to use only certain bits out of that how do we do that?

So example here, A is my 4 bit signal and my sum is 16 bit out of the 16 bit I want to select only some certain 4 bits. So what I can do is I can say for example 12 to 15 of this some I would like to assign to A, so I can say A equal to sum 15 colon 12. Now, again remember the same thing. So if I want to assign a 0 th bit of 8 to 15 bit of sum and first bit of A is equal to 14 bits, so then I can reverse I can write it in interchangeable manner I can write sum square bracket 12 colon 15.

So 12 will become more significant and 15 will become least significant and that would be assigned one to one you to this left hand side variable. So if I want to take only single bit out of it, so let us say I this T is a single bit wire then I can say any particular signal so let us say 15, so fifteenth bit of sum would be assigned to A. If two signals are of same size then no such square bit specification is required.

So you can simply say A equal to B, now A and B both are 4 bit of wide signal, so A could be assigned to B without writing anything in the square bracket. So, this is how we can do port selection.

(Refer Slide Time: 4:48)



Multiple bit width

- Concatenation operator
  - A = {a4,a3,a2,a1}
  - sum = {16{1'b1}
    //sum = 1111 1111 1111 1111
  - sum = {4{a4}, 4{a3}, 4{a2}, 4{a1}}
- Constant assignment
  - A = 4'b0001;
  - Sum = 16'h123C
- Display : %d decimal, %b binary, %h hex
  - $display("A: %b, B: %b, Sum: %h", A, B, sum);

Digital Logic Design:Combinational Circuits. 73

There is a reverse operation also that we can get in it, so you see from the previous declaration a1 a2 a3 a4, all of them were wires. Now, if I want to assign and group them together as a 4 bit signal, so I can put a square bracket, put a curly braces and put them curly brace is actually concatenation operator.

So in this curly braces plus with these comas whatever we write that would get combined. So when we are writing a4 a3 a2 a1, all of them get combined to become a. So, here most significant bit would be the first bit which we have written here and least significant would is the one which we have written in the end. So this concatenation operation can also be applicable in other ways where we can say something like this.

So I want to initialize I want to say that this pattern has to be repeated again and again, so this is a 1 b 1 so that means a 1 bit with value 1. I want to repeat it 16 times, so then also I can use catenation operator where I am saying that 16 time repeat this particular pattern. So this can be generally applicable, so instead of this there could be a variable name also or a signal name also. So, for example here, I can also say sum is equal to 4 times a4, 4 times a3, then 4 times a2, then 4 times a1.

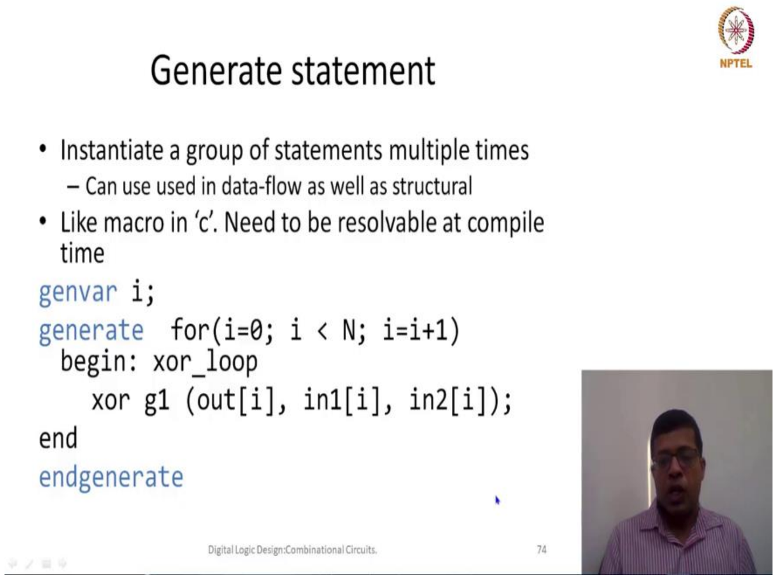
So, whatever is the value of a1 whether it is 0 or 1 it would be repeated 4 times. So that is how this catenation operator would also be required whenever we are handling this multiple bit width signals. Now, let us also quickly see how to write constants when the size of bits is more. So we have seen earlier so if we are writing for single bit we have to write 1b and then apostrophe and then value.

So, if I want to write like 4 bit, a is my 4 bit signal, so I can write 4 apostrophe b. So 4 apostrophe B and then I can write 0 0 0 1, so similarly if instead of binary I want to write in hexadecimal then instead of b, I can write h then you can say 16 apostrophe h and then whatever hex number I would like to assign I can write. Because sum is a 16 bit number so I have to write this 16.

So what would happen if left hand side bits and right hand side bits they did not match. So, then if left hand side, so if my left hand side is less than the rest of the bits would be assumed as 0 if this is more than the bits which are it in the higher significant here would be ignored. So, what about display so you did not want to display then we can use like in C, C plus plus we use the similar modifying operators can be used here.

So I can use percentage d for decimal, percentage b for binary and percentage h for hex. So for example, here I want to display a in binary, I want to display b in binary and I want to display sum in hex, so I can write correspondingly percentage b percentage b percentage x etcetera.

(Refer Slide Time: 8:26)



Generate statement

- Instantiate a group of statements multiple times
  - Can use used in data-flow as well as structural
- Like macro in 'c'. Need to be resolvable at compile time

```
genvar i;
generate  for(i=0; i < N; i=i+1)
  begin: xor_loop
     xor g1 (out[i], in1[i], in2[i]);
end
endgenerate
```

Digital Logic Design:Combinational Circuits. 74

So another very low construct which would be very helpful would be generate statement. So in the generate statement what we do, we instantiate a group of statements multiple times using some sort of a for loops, so it can be used in data flow as well as in structural way. So it is the generate segments are very much like a, c macros that means it has to be resolvable at compiled time so that means we have to give some constant values here and it would be inflated, inflated means it would be unrolled or it would be (at) before your execution, before your compilation.

So for example, so whenever we are using this generates statement, so there would be two constructs which are two constructs which are important one is genvar. So, genvar is the variable which would be the loop iterator and there would be a generate statement and within the generate statement usually there would be a for loop. So in this for loop I can say for i equal to 0, i less than N, i equal to i plus 1.

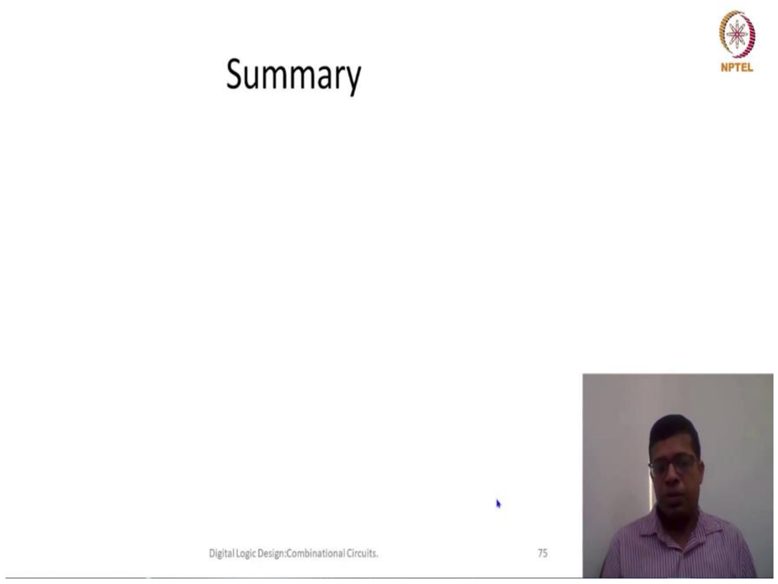
So, this i is a generate variable, so this variable would be fixed wherever in this for loop i would be there and all these instance would be created. So, for example in this example, what I want to create is I want to create an (and) N input XOR gate. So, because in each XOR gate I am saying that out i and input 1i, input 2i, so that means there would be n number of such gates would be generated and N has to be a constant, it has to be a fixed constant so that I can do a compile time analysis I can generate it at compile time.

So using this now there you can also see that after this for we have seeing begin and this end is for this begin so this for loop has a begin so this this is the end for this for loop and now for generate we have to finish it as a end generate. So this is generate block, in this generate block whatever is there inside it will get a replicated at the elaboration time or at the at the execution time, it would be elaborated.

So that means when we are doing this it is still doing, it is still generating things structurally. There would be n number of XOR gates that would be that we will be able to see in our VCD files or so in our hierarchy also we can see that there are there n XOR gates and each of these XOR gate would be taking different input based on this genvar i. So, this is more effective method some time to write a circuit where the circuit looks very similar.

So, for example in the ripple carry adder we have to use n different single bit adders, which has very similar definitions so we can use such kind of a construct if we like.

(Refer Slide Time: 11:48)



Summary

Digital Logic Design:Combinational Circuits. 75

So with this we would like to close today's lecture in summary we have seen how to design adders, we have also analyzed the adders how the delay is why or how to compute delays and how to optimize delays. So all this with this analysis we can also use this kind of analysis in other circuits also in a similar circuits also and along with that we have also seen how to write very low with bit vector (as a) in the programs. Thank you very much