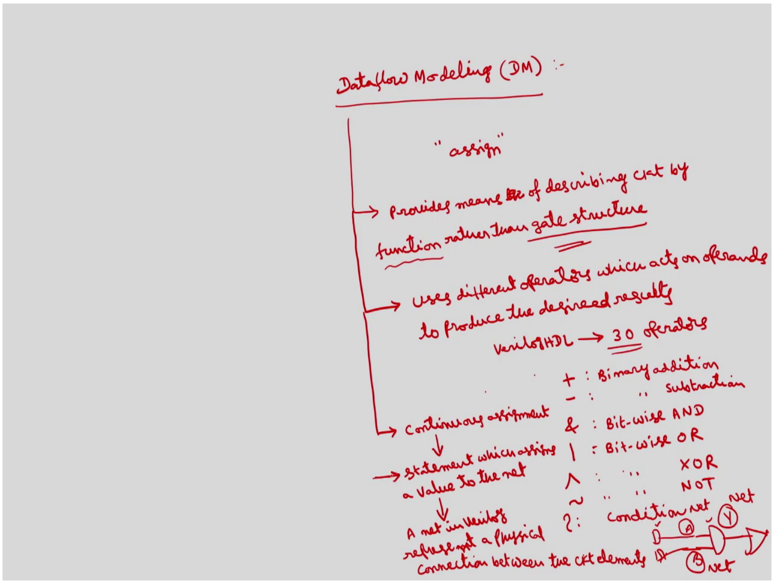
System Design using Verilog
Dr. Shaik Rafi Ahamed
Department of Electronics and Electrical Engineering
Indian Institute of Technology, Guwahati

Dataflow and behavioral modeling
Lecture - 14
Basics of dataflow modeling

In the last lectures, we are discussing about the gate level modeling or structural modeling or the switch level and transistor level modeling. So, normally this gate level modeling will be useful to describe the circuits consisting of fewer gates. For the large circuits or the complex circuits, which contains several gates; it becomes difficult to model using the gate level, so in that case we will go for this the data flow modeling.

(Refer Slide Time: 01:09)



Dataflow Modeling (DM) :-

"assign"

→ provides means of describing ckt by function rather than gate structure

→ uses different operators which acts on operands to produce the desired results

Verilog HDL → 30 operators

+ : Binary addition
− : " subtraction
& : Bit-wise AND
| : Bit-wise OR
^ : " XOR
~ : " NOT
?: condition

→ Continuous assignment
↓
→ statement which assigns a value to the net
↓
A net in Verilog represent a physical connection between the ckt elements

So, next step of the modeling is called data flow modeling called as DM. So, in case of gate level or structured modeling, we are going to use the key words corresponding to the gates. So, whereas, in data flow modeling, basically this data flow modeling uses a continuous assignment called as assign.

So, this data flow modeling basically provides means of describing the circuit by logic function, rather than gate structure. So, in case of gate level modeling, we are going to use the gate structure. So, this provides means of describing the circuit by means of a function, not the gate structure.

So, here how does this will describe these circuit. So, it basically uses the different operators. This dataflow modeling uses different operators, which acts on operands to produce the desired results. So, basically this uses the different operators. So, in Verilog HDL nearly 30 different operators are there, like this is binary addition; we have discussed these operators in the introduction class of this Verilog.

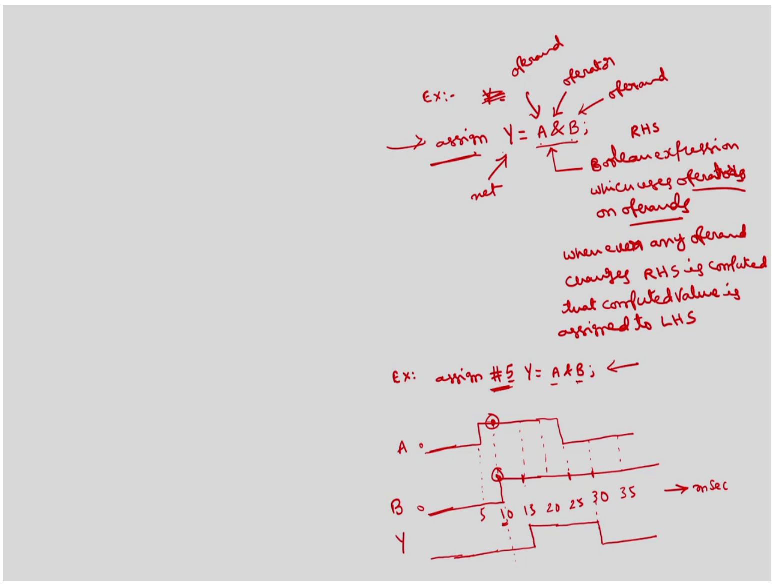
This is binary addition, similarly this is binary subtraction, this is bitwise AND, and this is bitwise OR, this is bitwise XOR, this is bitwise NOT, and this one is question mark followed by colon, this is nothing, but a condition. So, like this we have different operators, which are nearly 30.

So, how to express this assign statement? So, the assign statement uses continuous assignment, normally this data flow modeling uses continuous assignment. So, what is this continuous assignment? So, in continuous assignment, this is nothing, but a statement which assigns a value to the net. So, what is net? We have already described in the introductory class of this Verilog.

So, a net in Verilog represents a physical connection between the circuit elements. For example, if I take AND gate with the two inputs A, B, output Y; if this Y is connected to some OR gate, if these are coming from some other AND gates, this is coming from some exclusive OR gate.

So, the connection between the circuit element of this AND gate, and this AND gate is connected through these A. So, A is called as a net, similarly B will acts as a net, Y will acts as a net. So, basically a net is a physical connection between the circuit elements, ok. So, how does this statement is formed? See we are going to have some expression on the right hand side.

(Refer Slide Time: 07:01)


Ex:-
operand
operator
operand
assign Y = A & B;
RHS
net
Boolean expression which uses operators on operands
whenever any operand changes RHS is computed that computed value is assigned to LHS
Ex: assign #5 Y = A & B;
A
B
Y
5 10 15 20 25 30 35
→ msec

So, if I take the example of this assignment, assign Y is equal to A, AND gate is represented bitwise AND is by &. So, whenever this right hand side is a expression, this is called normally Boolean expression, which uses operators and operands. Here only one operator is there and operation; this is operator and A and B are called operands, this is one operand, this is another operand. So, basically this right hand side, RHS will be represented by a Boolean expression, which uses operators on operands.

So, whenever any operand changes, then the RHS is computed, that computed value is assigned to LHS left hand side, this is Y using this assign statement. So, sometimes to assign the computed value to the left hand side net, this is basically net, after some time. So, in that case we can use delay also.

So, another example of this assignment through delay is, suppose if I write assign; # is the symbol for the delay, which you have discussed in the earlier classes. #5 I want to assign after 5 milliseconds, Y = A & B. So, the difference between assign Y = A & B and assign #5 Y = A & B is, here we have this delay. So, here in this case whenever A changes or B changes; this AND operation will be completed and that will be assigned to Y.

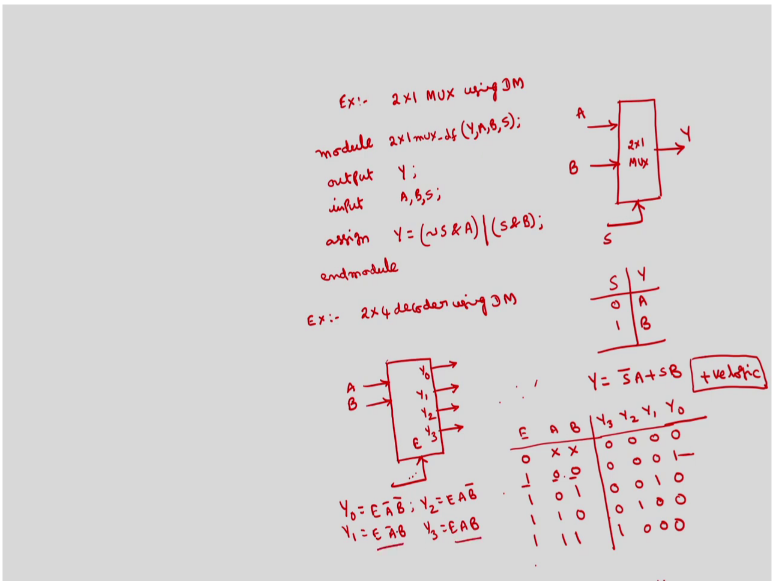
Whereas here whenever A changes or B changes; after 5 milliseconds, this A and B will be assign to Y. For example, if I take say A is something like this and B is something like this; if

I assume that this say 5, this is 10, 15, 20, 25, 30, 35 and so on, this is a millisecond. This is A, this is B; let us assume that they are at logic 0 from the infinity. Let us assume that they are coming from infinity with logic 0. Then, what will be Y?

So, initially because these two are 0 from infinite, Y is also 0; but at the point of 10, actually this is becoming 1, this is also already 1. So, output should be 1, because this is AND operation. But here because of these 5 seconds delay; this 1 will appear at 15 seconds, rather than 10 second. Here this becomes 1, actually this 1 has been produced at these 10 seconds only, 10 seconds B is also logic 1, A is also logic 1, output should be 1.

But because of this 5 milliseconds delay, the output Y becomes 1 at 15th second. Now, this will continue up to 25th second it should be 1, because both are 1s. So, after 25th second, output should go 0; but because of these 5 seconds delay, it will go 0 at 30th second. So, this will continue up to this one and at 30th second, it will go to the 0. So, this is the difference between the direct assign and assign with delays. So, basically in data flow modeling, we basically use the assign keyword.

(Refer Slide Time: 12:19)



Ex:- 2x1 MUX using DM

```
module 2x1mux_df (Y,A,B,S);
output  Y;
input   A,B,S;
assign  Y=(~S&A)|(S&B);
endmodule
```

S	Y
0	A
1	B

$Y = \bar{S}A + SB$ $\boxed{\text{+ve logic}}$

Ex:- 2x4 decoder using DM

$Y_0 = E\bar{A}\bar{B}$; $Y_2 = EA\bar{B}$

$Y_1 = E\bar{A}B$ $Y_3 = EAB$

E	A	B	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

So, I will give some other examples. So, I will write the data flow modeling of a 2 by 1 multiplexer; you have described this 2 by 1 multiplexer using the gate level modeling as well as circuit level modeling in the earlier lectures. So, we know that this 2 by 1 multiplexer will have two inputs; let us call the inputs as A, B and there will be a selection line S and there will be one output Y.

So, the operation is basically S, Y will be; S0 means y is A, S1 means Y is B. So, the Boolean expression for this one is. So, here we need the Boolean expression rather than a gate structure; as we have already discussed in case of dataflow modeling, we need a Boolean expression.

So, Y is given by S complement A + S B, ok. So, how to write these dataflow modeling? So, the initially we will define this as module; I will give the name as 2 by 1 mux under this one df, df stands for data flow modeling. There are three inputs and one output; input A and B, S and output Y. In case of gate level modeling, you have to write down the gate structure, so you have to implement this using basically gates.

Whereas in dataflow modeling, you have to write assign Y = (S complement & A) this is the bitwise operation. Then S AND B, then end module; this is relatively simple modeling when compared with the gate level modeling.

So, now I will take another example of decoder, 2 by 4 decoder, using data flow modeling. So, if I take the positive logic. So, this 2 by 4 decoder will be thing like two inputs A and B and we can have one more input called enable signal and four outputs Y 0, Y 1, Y 2, Y 3.

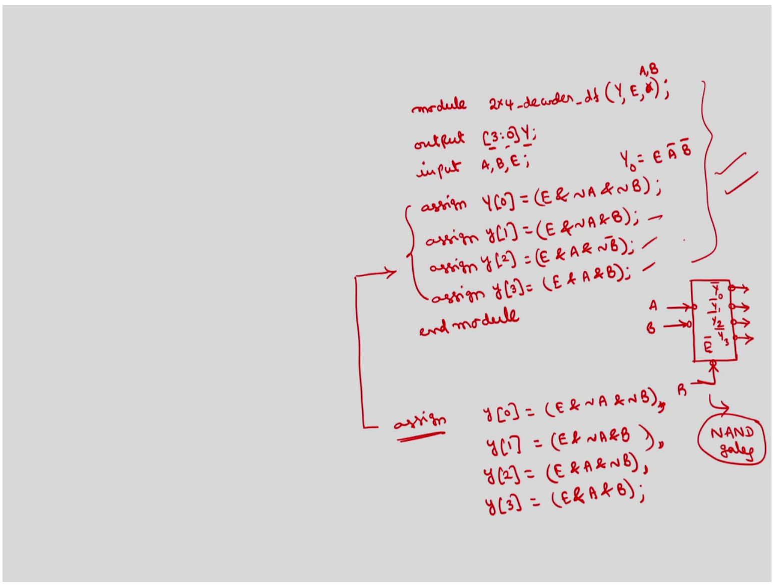
So, the operation is enabled signal, A, B; Y 3, Y 2, Y 1, Y 0. If enable is 0, regardless of A and B, none of the output will be selected; selected means 1, not selected means 0. So, in order to select any one of these outputs enable should be one. So, for 0 0, Y 0 is selected; for 0 1, Y 1 is selected; for 1 0, Y 2 is selected; for 1 1, Y 3 is selected, this is just positive logic.

We can design using negative logic also, most of these IC's that are available; use just negative logic, because negative logic circuit can be implemented by using the NAND gate, NAND gate is the universal gate. So, for the sake of simplicity here, I am using positive logic; this can be implemented by using AND gate.

So, what are the Boolean expressions for Y 0, Y 0 is 1 here. So, these two are 0s, but E should be 1. So, this is E, A bar, B bar; Y 1 is E is common in all the expressions A bar B; Y

2 is E, A B bar; Y 3 is E A B, ok. So, you can write down the Verilog code correspond to this decoder, basically here we have AND gates, ok.

(Refer Slide Time: 17:53)



```
module 2x4_decoder_df (Y, E, ~~X~~ A,B);
output [3:0] Y;
input A, B, E;                      Y0 = E A̅ B̅
assign Y[0] = (E & ~A & ~B);
assign y[1] = (E & ~A & B);
assign y[2] = (E & A & ~B);
assign y[3] = (E & A & B);
end module

assign  y[0] = (E & ~A & ~B),
        y[1] = (E & ~A & B ),
        y[2] = (E & A & ~B),
        y[3] = (E & A & B);
```

Module, so you have to give 2 by 4 decoder_df data flow modeling. So, basically we have four inputs and one enable input and four outputs, Y output and there are enable input and X is the another input; of course, X and Y are vectors here, later I am going to define that.

So, output 3: 0 Y. So, we have Y 3 to Y 0. Input two only, we have A B; this input is not X, this is A B. Say input A, B, E. Then we know that the expression for Y 0 is E, A bar, B bar, so assign Y 0; this is Y 0, because this is vector Y 3, Y 0, Y 1, Y 2.

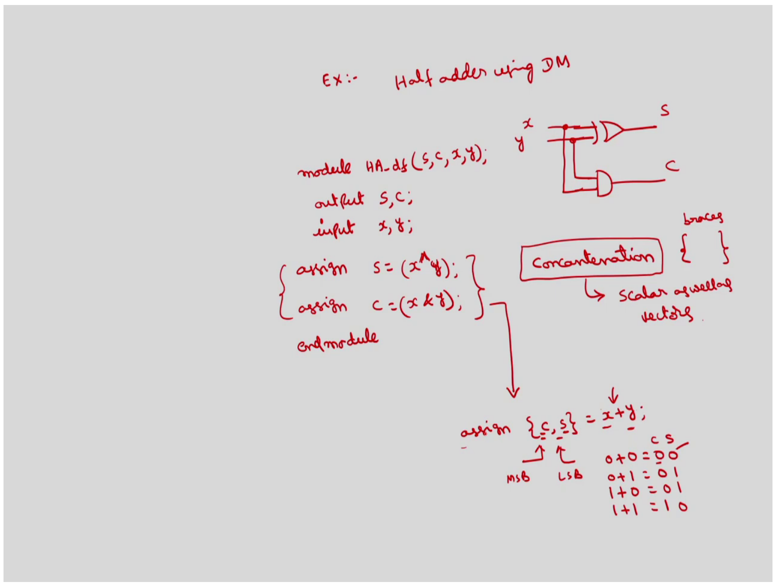
So, Y 0 is equal to E AND NOT operation with A AND NOT operation with B. Similarly, you can write assign Y 1 is equal to E AND with A bar AND with B; assign Y 2 is equal to E AND with A AND with B bar; assign Y 3 is equal to E AND with both A and B, end module. This is the dataflow modeling of a 2 by 4 decoder, if I use the positive logic.

Here you stop using these four assigns, we can write this alternatively as a single assign also. So, this you can replaced with a single assign; you can write Y 0 is equal to E AND with NOT A AND with NOT B. Then Y 1, for a single assign you have to put a comma here; Y 1 is equal to this expression, Y 2 is this comma and write otherwise E AND NOT A AND B, Y 2 is equal to E AND A AND NOT B, Y 3 is equal to E AND A AND B.

Now, you have to write the (Refer Time: 21:46) is another way to write this instead of using four assigns, you can use a single assign; in that case, you have to separate these using comma symbol. So, this is data flow modeling of a 2 by 4 decoder, which basically uses the AND gate. But normal decoders if I take, which are available in the IC form; if I take LS 74138, which is 3 to 8 decoders.

So, that the signals will be active low signals. So, here the signals A, B will be available as active low signals A, B, enable also active low signal, output also active low outputs. This will be available as Y 0 bar, Y 1 bar, this is also E bar, Y 2 bar, Y 3 bar. So, this type of circuit can be implemented by using NAND gates. So, similarly you can write this Verilog code for the negative logic circuit, which uses the NAND gates, ok. Then I will take another example of adder.

(Refer Slide Time: 23:12)


Ex:- Half adder using DM

```
module HA_df(s,c,x,y);
  output s,c;
  input x,y;
 assign s=(x^y);
 assign c=(x&y);
endmodule
```

Concantenation → scalar as well as vectors

braces { }

```
assign {c,s} = x+y;
        ↑ ↑
      MSB LSB
```

	c s
0+0 =	0 0
0+1 =	0 1
1+0 =	0 1
1+1 =	1 0

I will start with the half adder, I will go up to the look ahead carry adder. So, if I take half adder using data flow modeling. So, we know that half adder circuit will have two inputs x and y and two output sum and carry; this is sum, this is carry. So, module half adder_data flow s, c, x, y; output s, c, input x, y.

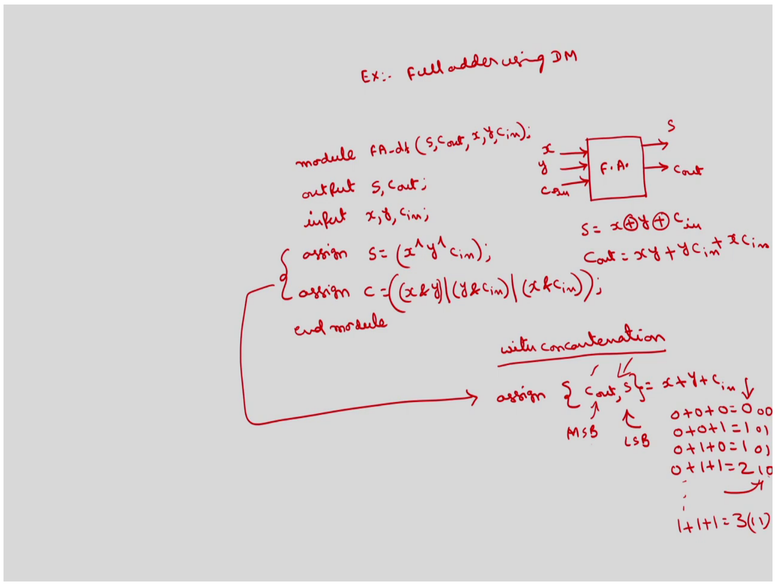
Assign s is equal to x exclusive OR symbol is ^ operator y; assign, you can use a single assign, you can put a comma also, carry is equal to x AND y. This is one way to write this Verilog code in data flow modeling. So, another concept is we can have concatenate; this is principle of concatenation, this can be applied on scalars as well as vectors.

If you want to use this concatenation so, this instead of having two assigns; so we can replace these two assigns with single assign and this concatenation has to be represented by braces; output is carry and sum, is equal to the inputs are x plus y. So, this will assign, this will basically perform the binary addition, this is an addition; here we do not use these symbols like binary operators.

So, this is binary addition basically, it will perform all the combinations here 0 plus 0, result is 0. So, 0 plus 1, this is if I want to write 2 bit equivalent of this one, this is 0 0, sum and carry. And if I write this carry and sum in the same order, 0 plus 1 is sum is 1, carry is 0; 1 plus 0 is carry is 0, sum is 1; 1 plus 1 is carry is 1, sum is 0. So, in the order that they will generate, this is MSB, this is LSB.

So, after this addition, whatever this LSB bit is assigned to this sum and MSB bit assigned to c. This is what the concept of concatenation; we can apply for scalars as well as the vectors. So, this is your x and y are scalars, they are single bit numbers; later we are going to apply the same principle on the parallel adders also, ok. So, you see about the half adder, you can write in two ways; you can use either assign or you can concatenate using a single assign.

(Refer Slide Time: 27:39)



Ex:- Full adder using DM

```
module FA-df (s,Cout, x,y,Cin);
output s, Cout;
input x,y, Cin;
assign s= (x^y^Cin);
assign c =((x&y)|(y&Cin)|(x&Cin));
end module
```

$S = x \oplus y \oplus C_{in}$

$C_{out} = xy + yC_{in} + xC_{in}$

with concatenation

```
assign {Cout, s}= x+y+Cin
```

x
y
Cin
F.A.
s
Cout
MSB
LSB
0+0+0=0 00
0+0+1=1 01
0+1+0=1 01
0+1+1=2 10
1+1+1=3(11)

Then full adder, I will write directly concatenation. So, the full adder is basically we have three inputs, this is full adder. Here in case of data flow modeling, we do not need the internal circuitry; if I know the function that is enough. So, these basically we have three inputs; this is x, y, cin if I call c suffix in, sum and carry out two outputs.

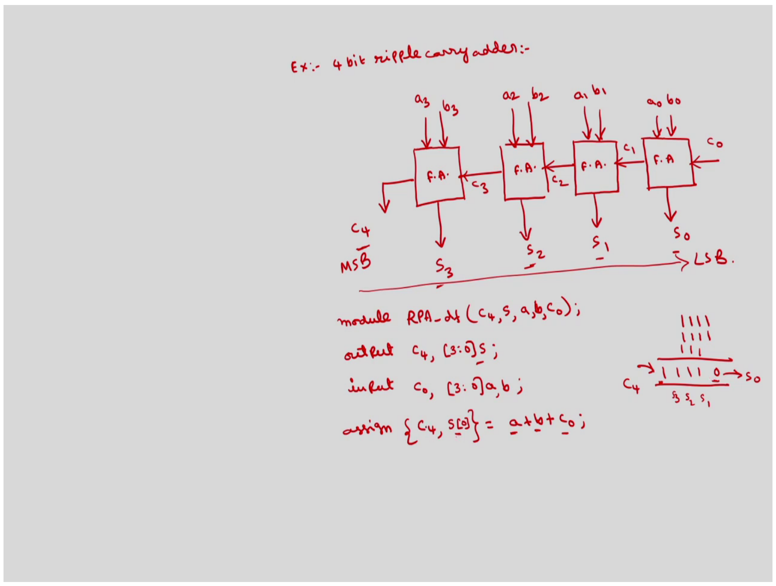
So, the expression for sum is x exclusive OR y exclusive OR cin; carry out is x y plus y c in plus x cin. We can write in two ways, similar to the previous half adder; module full adder_data flow modeling, you have c, c out, x, y, cin; output s, cout; input x comma y comma cin. First, I will write using the multiple assign statements; assign sum is equal to x exclusive OR y exclusive OR cin; assign carry is equal to x AND y OR y AND cin OR x AND cin.

You can write overall another bracket this, then end module. This is without concatenation. Now, if I use concatenation. So, these two assign statements can be changed to single assign statement. So, MSB here is carry, cout comma s is equal to x plus y plus cin; basically, this will perform the addition of the 3 bits. So, in that MSB bit we will assign to cout, LSB will be assigned to s; this will take all the eight combinations, the MSB bit. So, 0 plus 0 plus 0 is decimal 0.

So, 0 plus 0 plus 1 is the decimal 1; 0 plus 1 plus 0 is also decimal 1; 0 plus 1 plus 1 is decimal 2. So, these decimal values will be represented by 2 bit binary equivalent; this is correspond to 0 0, this is correspond to 0 1, this is correspond to 0 1, this is 1 0. So, in that this MSB bit you have assigned to cout and LSB is assigned to sum.

So, we know that in case of 0 plus 1 plus 1, the sum bit is 0, carry bit is 1. So, like that 1 plus 1 plus 1 is 3 in case of decimal; the binary equivalent is 1 1. So, both will be assigned 1 1. So, like that this concatenation of these vectors or scalars will be very much useful in the data flow modeling to simplify the code. So, now the next one is, we have 4-bit ripple carry adder.

(Refer Slide Time: 32:21)



Ex:- 4 bit ripple carry adder:-

```
module RPA_df (c4, s, a, b, c0);
output c4, [3:0] s;
input c0, [3:0] a, b;
assign {c4, s[0]} = a+b+c0;
```

If I want to write the codes using the gate level modeling, you have to instantiate the previous programs; like you write the first code for half adder, you instantiate twice that in the full adder, and you have to instantiate four times that full adder into the 4-bit ripple carry adder, that we have discussed in the earlier lectures. Whereas, here that is not required.

So, we know that in case of 4-bit ripple carry adder, we have four full adders. So, we know that four full adder we have three inputs and two outputs; these are this LSB inputs, let us call this one as a0, b0 and this is a c0, this is c1, this is c2, this is c3, this is c4, we can also called as c out. This will give sum bit s0, s1, s2, s3.

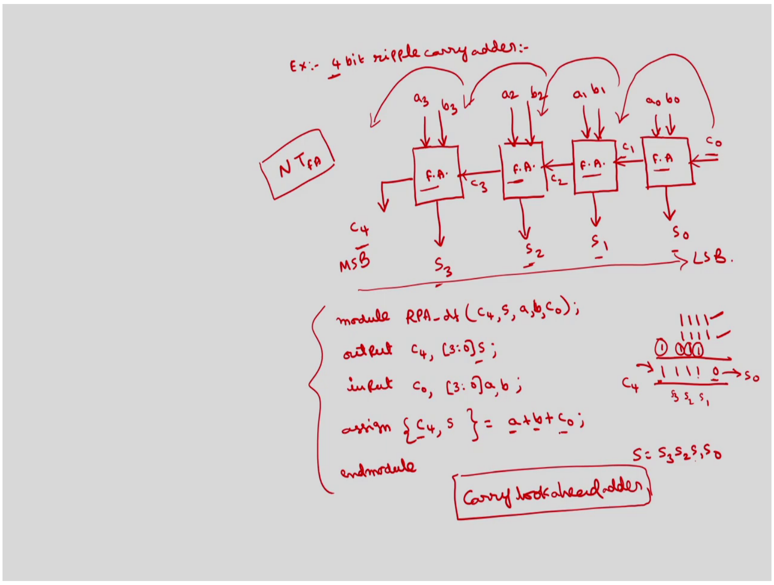
This is a1 b1, a2 b2, a3 b3; this is full carry adder. So, in order to write this code, we can use the concatenation principle. So, module ripple carry adder_ data flow modeling. So, the outputs are c4, s; inputs are a, b, c0; of course, the a, b and sum are here vectors.

Output c4, [3:0] s; this is s3, s 2, s 1, s 0; input c0, [3:0] a and b; both a, b are 4 bit numbers. I will write the little this concatenation of this outputs assign. So, what is the MSB bit? Total result is 5 bits. So, in this 5 bit result, this is LSB and this c4 is MSB.

If you add two 4 bit numbers, so the resultant can be 5 bits; because we are adding two 4 bit numbers, the maximum number is 1 1 1 1, 1 1 1 1. If we perform this addition 1 plus 1 is 1 0; 1 plus 1 plus 1 is 1 1; 1 plus 1 plus 1 is 1 1; 1 plus 1 plus 1 is 1 1. So, we will get at the most 5 bits. So, this represents the c4, this represents s0; this is s 1, s 2, s 3 this is s 0, s 1, s 3 and this is c4, ok.

So, in this MSB c4, assign c4 comma LSB is s0 or you can simply write s also s 0. So, what is that is simply a plus b plus c0. So, this a will take 4-bit number, this will take 4 bit number, this is 1 bit number. So, then among these 5 bits, s0 will be assigned; this least significant bit is assigned to s, we can also simply write s also here, this represents already inside this s.

(Refer Slide Time: 37:16)



```
module RPA_df (c4, s, a, b, c0);
output c4, [3:0] s ;
input c0, [3:0] a, b ;
assign {c4, s } = a+b+c0;
endmodule
```

So, s is nothing, but s3, s2, s1, s0. So, LSB bit is assigned to LSB, the next LSB is assigned to s1 so on, MSB is assigned to c4. Then we have end module. This is the beauty of the concatenation. And we can extend this to look ahead carry adder. So, as the number of stages increases if we want to add two 4-bit numbers; what is the maximum propagation delay?

Because this carry has to ripple through the adders, that is why the name ripple carry adder. So, from c 0, c 1 is generated; c 1 again it will propagate to c 2 in a ripple manner, that is why the name ripple carry. So, what is the total propagation delay to get the final output? If I take the same in the worst-case principle of this 1 1 1 1; here we will get a carry, here we will get a carry, here we will get a carry, here we will get a carry.

This is the worst-case propagation delay is four times the propagation delay of each full adder. If I want in general n bit ripple carry adder, the total worst-case propagation delay is N times the propagation delay of full adder. As the number of stages increases, the propagation delay also increases.

So, to avoid these regardless of the number of stages; if you want to have a constant propagation delay, we have to go for carry lookahead adder. So, we will discuss this carry look ahead adder in the next lecture and we will write down this dataflow modeling using this concatenated principle.

Thank you.