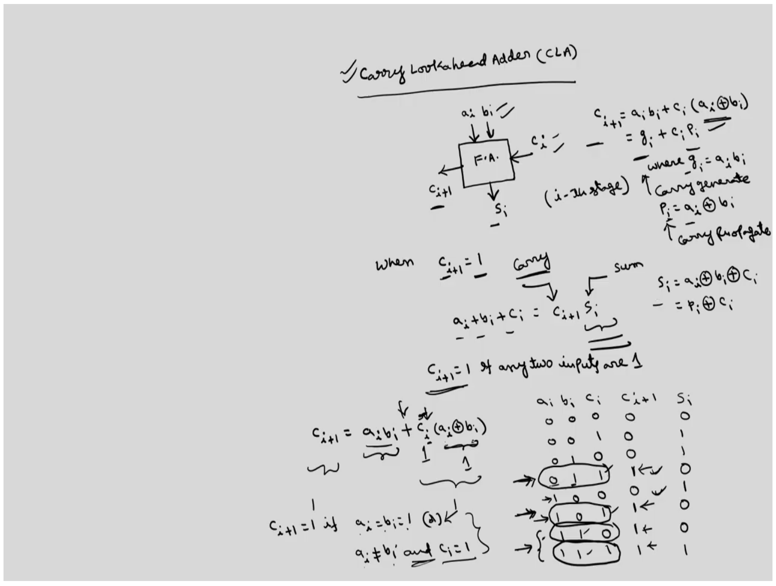
System Design Through Verilog

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

Dataflow and behavioral modeling
Lecture - 15
Examples of dataflow modeling

In the last lecture, we are discussing about the Dataflow Modeling. So, we have discussed about the dataflow modeling of some combinational circuit such as multiplexer, decoder, then half adder, full adder, ripple carry adder.

(Refer Slide Time: 00:57)



Carry Lookahead Adder (CLA)

$c_{i+1} = a_ib_i + c_i(a_i \oplus b_i)$
$= g_i + c_ip_i$

where $g_i = a_ib_i$ ← carry generate
$p_i = a_i \oplus b_i$ ← carry propagate

(i-th stage)

When $c_{i+1} = 1$

$a_i + b_i + c_i = c_{i+1}s_i$

$s_i = a_i \oplus b_i \oplus c_i$
$= p_i \oplus c_i$

$c_{i+1} = 1$ if any two inputs are 1

$c_{i+1} = a_ib_i + c_i(a_i \oplus b_i)$

$c_{i+1} = 1$ if $a_i = b_i = 1$ (a)
$a_i \neq b_i$ and $c_i = 1$

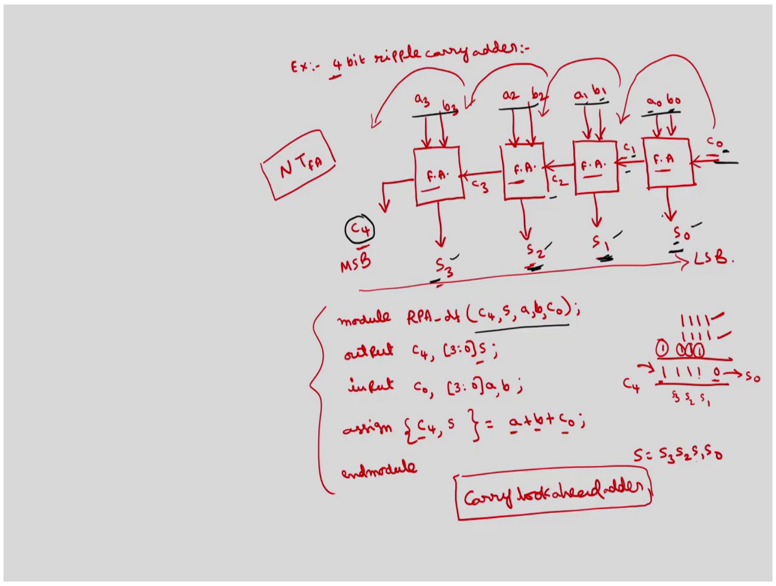
a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

So, the next adder is Carry Lookahead Adder, CLA. So, the drawback of the ripple carry adder is the carry will propagate through the stages, as a result of that the total propagation delay depends upon the number of stages. In general, if you want to add 2 n bit numbers, then the total propagation delay is n times that of the full adder.

So, it is the drawback of this ripple carry adder. To avoid that, we can use this carry lookahead adder ok, where regardless of the number of stages, the delay caused for the performing the addition will be same.

So, what is the idea behind this carry lookahead adder? So, if I take any ith stage and i+1 stage, this is typically a fully adder and we are considering this as ith stage a_i, b_i, c_i and output will be c_i plus 1; in this output is s_i. This is a typical ith stage. We can see from this the previous this one also.

(Refer Slide Time: 02:48)



```
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, here, you can see here for the zeroth stage these are 0, a_0, b_0, output is s_0, input is c_0, output is c_1. For the first stage, these are 1 1 1 1 and this is 2. So, like that if I take the ith stage, this is c_i, a_i, b_i, s_i, c_{i+1}. Now, here as the name implies, we are going to generate all the carrys at a time ok, instead of generating this c_1 first and then c_2 and then c_3 and so on, we will generate all the carries simultaneously.

So, in order to generate these carries simultaneously, so what is the logic that we have to follow. If I consider this ith stage only, so when does this c_{i+1} becomes 1? This is basically adding three numbers, $a_i + b_i + c_i$. So, this will give 2 bits sum and carry; this will give c_{i+1}, s_i. This is sum bit and this is carry bit.

So, we are not interested in the sum bit. Let us forget about this. So, when does this carry bit will be 1; this c_{i+1}? c_{i+1} is equal to 1, if any two bits are 1, so among these three, at least two bits has to be 1. So, among all the eight combinations, what are the combinations? So, this is

a_i, b_i, c_i, if I take all the combinations 0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1. Among all these combinations, so what is this sum c_{i+1}; when will this be 1?

So, 0 plus 0 plus 0 is 0, there is no carry; 0 plus 0 plus 1 is 1; sum bit is 1, carry is 0. This is also sum bit is 1, carry bit is 0. Here, we will get carry bit 1, sum bit is of course 0. If you want write this sum also, we are not interested here; but I will write this. So, this is 0, this is 1, this is 1, this is 0 and here, also because only 1 input is 1, carry is 0, but sum is 1. Here 2 inputs are 1, so carry is 1, sum is 0. Here also carry is 1, sum is 0. Here both will be 1. This is the truth table of a full adder right.

So, the condition for c_{i+1} to be 1 is so here at least two inputs has to be 1. This is 1, this is the combination, where this outputs c_{i+1} is 1. This is another combination; this is another combination. So, out of these eight combinations, in four combinations, c_{i+1} is equal to 1.

So, in order to represent this using, the Boolean expression, c_{i+1} is equal to we can derive in terms of a, b, c. But I am going to derive in terms of exclusive OR and AND operation so that I can define some terms like carry propagate and carry generate. So, one condition is if a and b if I assume that both a and b's are 1, I can write this a_i, b_i.

So, if both are ones, then only the c_{i+1} is 1. If anyone of this input is 0, c_{i+1} is 0. If I write only this condition a_i, b_i so this will cover these two combinations. These two combinations will be covered by simply a_i, b_i because in these two combinations, both a_i, b_i are 1; but there are two more combinations in which a_i, b_i are not 1, still c_{i+1} is 1.

So, how to cover these 2? If I write simply c_{i+1} is equal to a_ib_i, it will cover only these two combinations; but I have to cover these two combinations also. So, for that, I have to add some more extra term plus. So, what are these two combinations in which a_i is not equal to $b_{i.}$

So, a_i is not equal to b_i, still c_i is equal to 1. This can be represented by c_i (a_i exclusive or with b_i). So, in these two combinations a_i is not equal to b_i, so a_i exclusive or with this 1, so one of this one is 0, so this will be 1. This operation is 1 and if c_i is equal to 1, then this product term is equal to 1. If c_i is also 1, this product term is 1; thereby, this c_{i+1} is equal to 1. So, what are the two conditions under which the c_{i+1} is generated is if a_i, b_i both are 1, then is one condition or if a_i is not equal to b_i and c_i is equal to 1.

So, if I want to write in the words, statements, we can write c_{i+1} is equal to 1, if a_i is equal to b_i is equal to 1 or a_i is not equal to b_i and c_i is equal to 1. So, this statement can be mapped onto the Boolean expression using this, a_i is equal b_i is equal to 1 is a_i b_i; a_i is not equal to b_i is a_i exclusive OR b_i; c_i is equal to 1 into c_i and this AND is there, this is AND operation here. This is OR is there, this OR operation.

So, this expression for the c_{i+1} is equal to a_i b_i + c_i (a_i exclusive OR b_i). So, I am going to define this as g_i as b_i AND b_i plus c_i p_i, where g_i is a_i b_i is called as carry generate. So, this will generate the carry and p_i is a_i exclusive OR b_i, this is called carry propagate.

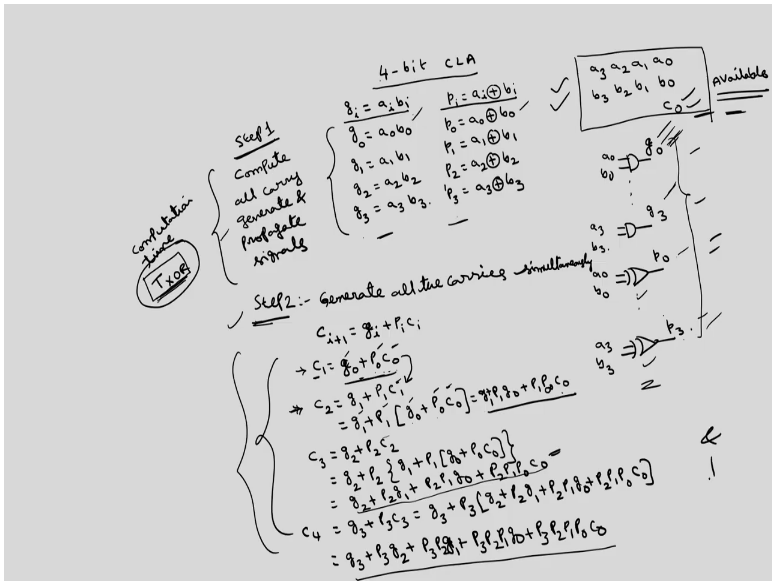
So, this is carry generate and this is carry propagate because this is going to propagate the carry of this c_i. So, in terms of g_i and p_i, c_{i+1} is g_i plus this. So, I can generate the c_{i+1} from $a_{i,}$ $b_{i,}$ c_i, then what about s_i? So, for a fully adder, what is the expression for the sum? s_i is nothing but exclusive or of all the three inputs; a_i exclusive or b_i exclusive or c_i.

But a_i exclusive OR b_i we are calling as propagate signal. So, you can simply call this one as p_i exclusive OR with c_i. So, in order to design this carry lookahead adder, the very first step is you have to generate all the generate signals and propagate signals given a’s and b’s and c’s, first we have to generate g’s and p’s.

So, after that, you have to find out all the carrys, then after that we will find out the sum. So, there are three steps in the carry look ahead adder. The first step is you have to produce all the generate signals and propagate signals, using some logic. Then, the second step is using these generate and propagate signals, you have to generate all the carrys simultaneously.

The third step is using these carries and this c_i, we have to generate all the sums. So, we will discuss now how to generate this all the carrys simultaneously.

(Refer Slide Time: 12:47)


4-bit CLA
$g_i = a_i b_i$
$g_0 = a_0 b_0$
$g_1 = a_1 b_1$
$g_2 = a_2 b_2$
$g_3 = a_3 b_3$.
$p_i = a_i \oplus b_i$
$p_0 = a_0 \oplus b_0$
$p_1 = a_1 \oplus b_1$
$p_2 = a_2 \oplus b_2$
$p_3 = a_3 \oplus b_3$
$a_3\ a_2\ a_1\ a_0$
$b_3\ b_2\ b_1\ b_0$
c_0
Available
Step 1
Compute all carry generate & propagate signals
Computation time
T_{XOR}
g_0
g_3
p_0
p_3
Step2:- Generate all the carries simultaneously
$c_{i+1} = g_i + p_i c_i$
$c_1 = g_0 + p_0 c_0$
$c_2 = g_1 + p_1 c_1 = g_1 + p_1 [g_0 + p_0 c_0] = g_1 + p_1 g_0 + p_1 p_0 c_0$
$c_3 = g_2 + p_2 c_2 = g_2 + p_2 \{g_1 + p_1 [g_0 + p_0 c_0]\} = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0$
$c_4 = g_3 + p_3 c_3 = g_3 + p_3 [g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0]$
$= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$
&

So, if I consider these 4-bit lookahead carry adder, CLA, so the input is first 4-bit number a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0 and also, will be given the input carry also c_0, these are available. So, very first step is here you have to generate all the propagate and generate signals. So, what is g_0? In general, g_i is $a_i b_i$; $a_i b_i$ is g_i and a_i exclusive OR b_i is p_i. So, g_0 is $a_0 b_0$; g_1 is $a_1 b_1$; g_2 is $a_2 b_2$; g_3 is $a_3 b_3$ and then, propagate signal p_i is equal to a_i exclusive OR b_i.

So, what is p_0? It is a_0 exclusive OR with b_0; p_1 is equal to a_1 exclusive OR with b_1; p_2, a_2 exclusive OR with b_2; p_3, a_3 exclusive OR with b_3. The first step is computing carry generate and propagate signals. This is your step 1. So, regardless of the number of bits, we have only three steps unlike ripple carry adder. In case of CLA, we have only three steps only.

The first step is we will generate all the generate signals and propagate signals. If it is 5-bit, we will generate g_4 and p_4 also. If it is 6-bit, we will generate g_5, p_5 also. So, for this what is the circuitry? We require AND gate. All these will be generated simultaneously because these are available.

So, this all g signal can be generated using AND gate. This will be generated by exclusive OR gate. Of course, the delay of exclusive OR gate is more than that of AND gate. So, what is the computation time to perform this step 1? The computation time for this one is T_{XOR} gate because parallelly all will be generated. So, this is a_0 b_0 because these are all available, this will generate g_0. Similarly, we have g_3, using a_3b_3, a_0b_0 will get p_0 so on up to a_3b_3, we will get p_3.

So, all these will be generated parallelly; but because this delay time is more, after this time of T_{XOR} gate. So, all g's and p's will be available. Of course, these g's will be available before the p's are available. So, if I wait up to T_{XOR} because without generating the p's, I cannot generate the all carries. We are going to follow the second step.

In second step, we are going to use all g's and p's. Even though, if generate this g's a little bit faster than this p's, so you have to wait until these p's are generated. So, after a time of T_{XOR} seconds, all the p's and g's will be generated. Now, we will proceed for the second step, generate all the carries.

Now, what are the inputs for this carry generator? We have g's and p's, even we have c_0 also. So, what is the expression for c_{i+1} is equal to $c_i + p_i\, c_i$. So, c_0 is the input which is given here available. So, you find out c_1 first. How to generate c_1. So, i is 0, so $c_0 + p_0$, this is $g_i + p_i\, c_i$; $g_0 + p_0\, c_0$.

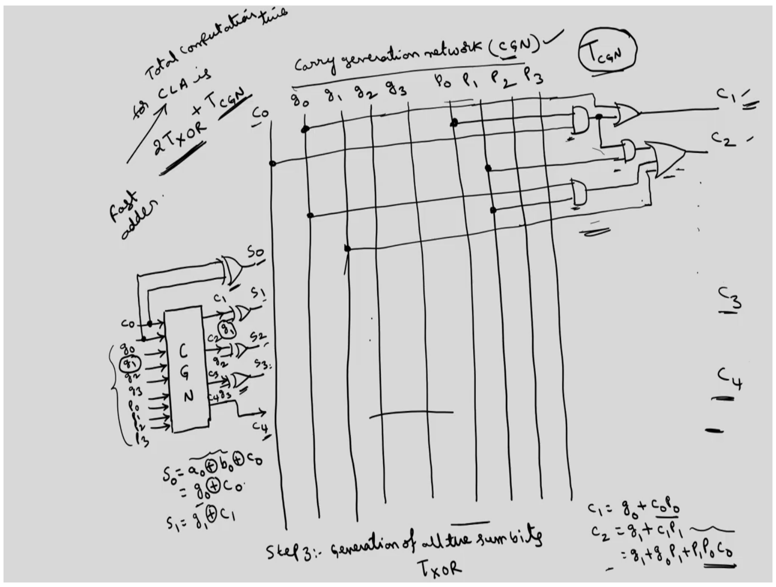
So, this g_0, p_0 are generated in the first step and c_0 is already available along with the a's and b's. So, I can generate this c_1 after some time. What is the time taken to generate this c_1 is one AND operation, one OR operation. Now, c_2 is given by $g_1 + p_1\, c_1$. If I compute this c_2 using this expression, then the c_1, so after generating the c_1 only, I can compute the c_2.

But here in this step, I want to generate all the carries simultaneously. For that, you have to substitute this c_1 here; $g_1 + p_1$. What is c_1? $g_0 + p_0\, c_0$. So, that we know this p_1, g_1, g_0, p_0 which are generated in the first step and c_0 is available at the starting of this design. So, we can generate c_2 after a time of this logic gates. So, c_2 will be generated after time of the time taken to perform these all Boolean operations.

Then, c_3 is $g_2 + p_2\, c_2$, this is equal to $g_2 + p_2\, c_2$ is nothing but we have to substitute this $p_2\,(g_1 + p_1\,(g_0 + p_0c_0))$. You can further simplify this expression so that you will get the simplified circuit. So, this is equal to $g_1 + p_1g_0 + p_1\, p_0\, c_0$ and this will be $g_2 + p_2\, g_1 + p_2\, p_1\, g_0 + p_2\, p_1\, p_0\, c_0$; then finally, the c_{out} c_4 is the last output. So, this is given by $g_3 + p_3\, c_3$, this is equal to $g_3 + p_3$; c_3 is from the previous expression, $g_2 + p_2\, g_1 + p_2\, p_1\, g_0 + p_2\, p_1\, p_0\, c_0$.

So, finally, we will get $g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0$. So, we are going to generate all the carries; c_0 is given as input carry, then we will generate c_1, c_2, c_3, c_4 using these Boolean expressions; this is the second step. So, what is the circuitry required for this one? So, this depends upon only c_0, p's and g's. We have generated four p's, four g's and then c_0.

(Refer Slide Time: 22:19)



Total computation time for CLA is
$2T_{XOR} + T_{CGN}$
fast adder.
Carry generation network (CGN)
T_{CGN}
c_0
g_0 g_1 g_2 g_3 p_0 p_1 p_2 p_3
c_1
c_2
c_3
c_4
S_0
S_1
S_2
S_3
C G N
$S_0 = a_0 \oplus b_0 \oplus c_0$
$= g_0 \oplus c_0$
$S_1 = g_1 \oplus c_1$
Step 3:- Generation of all the sum bits
T_{XOR}
$c_1 = g_0 + c_0 p_0$
$c_2 = g_1 + c_1 p_1$
$= g_1 + g_0 p_1 + p_1 p_0 c_0$

Carry generation network, if I call as CGN. So, basically, we have nine inputs; out of which this is c_0 line, this entire line is vertical line is c_0 and this is g_0, we have generated in step 1; g_1, g_2, g_3 you have generated in step 1. Similarly, p_0, p_1, p_2, p_3 you have generated in the step 1. So, what is the expression for c_1? We have derived c_1 as $g_0 + c_0 p_0$.

So, this was the expression, we have derived; $g_0 + p_0 c_0$. So, simply this requires $c_0 p_0$ AND gate. This dot represents a connection. This is $c_0 p_0$ and then, you have to add with g_0. This is g_0. So, this will generate your c_1.

Similarly, what is the expression for c_2 is $g_1 + c_1 p_1$. This is equal to if we substitute this c_1 here, $g_1 + g_0 p_1 + p_1 p_0 c_0$. So, we can have this previous one is we have $c_1 p_1$. So, $c_0 p_0$ is there, this term is already there here. So, we can take that term. This term is nothing but $p_0 c_0$, that you have to AND with p_1, so that this entire term is generated. Then, $g_0 p_1$; second AND term is generated, then g_1.

So, these three if you OR we will get c_2. Similarly, you can generate c_3 and c_4 using logic gates. See here, among all the paths, the path which takes the maximum computation time is the time taken to generate all the carries; C_{max} if I call as for example, if I take between these two, which one is the longest path? This is one OR gate followed by an OR gate, AND gate followed by OR gate, this is also another AND gate, another AND gate, another OR gate. So, this path is taking more time.

So, like that, if you take the circuit diagram of c_3, c_4 also. So, among all the paths, the maximum time taken by the signal to propagate from this c_0, g's and p's to c_1, c_2, c_3, c_4 that computation if I call as c_{max}. So, this c_{max} is the maximum computation time required to generate all the carries. So, in the first step, what is the computation time is T_{XOR}. Here also we call as T_{max}; instead of C_{max}, T_{max} or you can also call as otherwise because you are calling this as CGN, you can call as T_{CGN}.

Now, you have generated all the carries and the last step is once if we know the carries, I will represent this entire circuitry in a block. This I will call as CGN. What are the inputs and outputs of CGN? We have four p's, four g's and one c_0; c_0, g_0, g_1, g_2, g_3, p_0, p_1, p_2, p_3 and what are the outputs? c_1, c_2, c_3, c_4.

So, after the first step, using this AND gate and exclusive OR gate, we have generated all p's and g's. Second step using this carry generation network, we have generated c_1, c_2, c_3, c_4. So, what is the last step? We have to generate the final sum and final carry c_4; c_4 is the final carry of course.

So, what about s_0, s_1, s_2, s_3. As I have defined s_0 is a_0 exclusive OR with b_0 exclusive OR with c_0; this is nothing but g_0 exclusive OR with c_0. So, this you can take from here itself. This if you exclusive OR because a_0 b_0 is exclusive OR of g_0. So, this is exclusive OR this will give sum bit s_0. Similarly, s_1 is g_1 exclusive OR with c_1. So, c_1 is here. So, you exclusive OR with this g_1. This g_1 is nothing but this g_1. I have not shown the connection. This will give s_1.

Similarly, c_2 if you exclusive OR with g_2, we will get s_2; c_3 exclusive OR with g_3, we will get s_3 and this is c_4 is the final carry output . So, this is like c_4 is the final output; sum bits are s_3, s_2, s_1 s_0 .

So, here also we have generated these in three steps. We have generated s_0, s_1, s_2, s_3 and then, c_4 is the final output. Third step is generation of all these sum bits. So, what is the time taken to generate all these sum bits? Only exclusive OR operation. So, this is another T_{XOR}.

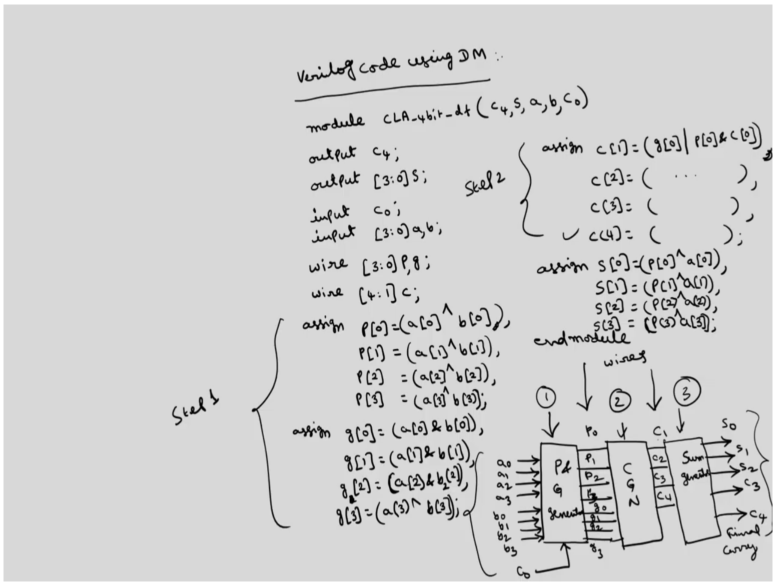
So, the first step also we will take T_{XOR} time to generate all g's and p's; second step we will take T_{CGN}, this CGN depends upon this Boolean expression that we have derived; this Boolean expression. To implement this Boolean expression, what is the maximum time taken? That will be T_{CGN} and to generate the last step, third step is T_{XOR}.

So, the total computation time of this CLA is $2T_{XOR}$ if first and third step requires the same time $T_{XOR} + T_{CGN}$, the time taken to compute this carry generation network. So, this is fixed regardless of the number of bits.

You can take 10-bit addition, you can take 20-bit addition, you can take 100-bit addition. This is same; whereas, this is slightly there will be increase in this T_{CGN}, if we increase the number of bits. So, in that way, we can reduce this carry propagation time. As a result of that, this carry lookahead adder also can be called as fast adder.

Now, if you want to realize or if you want to model this carry lookahead adder using gate level modeling, it becomes very difficult because here we have a lot many gates. But using this dataflow modeling, we can easily write the VERILOG code. Now what is the VERILOG code using dataflow modeling?

(Refer Slide Time: 33:07)



Verilog code using DM :

```
module CLA_4bit_df (c4, s, a, b, c0)
output c4;
output [3:0] s;
input c0;
input [3:0] a, b;
wire [3:0] p, g;
wire [4:1] c;
assign p[0] = (a[0] ^ b[0]),
       p[1] = (a[1] ^ b[1]),
       p[2] = (a[2] ^ b[2]),
       p[3] = (a[3] ^ b[3]);
assign g[0] = (a[0] & b[0]),
       g[1] = (a[1] & b[1]),
       g[2] = (a[2] & b[2]),
       g[3] = (a[3] ^ b[3]);
assign c[1] = (g[0] | p[0] & c[0]),
       c[2] = (   ...   ),
       c[3] = (         ),
       c[4] = (         );
assign s[0] = (p[0] ^ a[0]),
       s[1] = (p[1] ^ a[1]),
       s[2] = (p[2] ^ a[2]),
       s[3] = (p[3] ^ a[3]);
endmodule
```

Step 1
Step 2
wires
1
2
3
a0
a1
a2
a3
b0
b1
b2
b3
c0
P & G generator
p0
p1
p2
p3
g0
g1
g2
g3
C G N
c1
c2
c3
c4
Sum generator
s0
s1
s2
s3
c4
final carry

So, I will write module CLA 4-bit dataflow, I will give the name as CLA_ 4-bit df. So, what are the total inputs and outputs? We have c_4 final output and then, we have four sum bits, sum s, then we have four a’s, four b’s, c_0 is the input carry. This we have already used it in the dataflow modeling of lookahead carry, here we have used these.

This a’s, b’s are available as the inputs and c_0 is available; outputs are s_3, s_2, s_1, s_0 and final carry. Then, you can define the values like output c_4 a scalar. So, I am writing separately, you can combine with vector also and other outputs are [3 : 0] s because we have s_0, s_1, s_2, s_3.

Similarly, input c_0 is a scalar and another vector inputs are we have [3 : 0] a , b; both a and b are 4-bit numbers. Then, we require some wires also. So, what are the wires required? You see here the first step is generation of g’s and p’s using this exclusive OR gates. Second step is using these Boolean expressions will generate all the carries, third step is using ‘exclusive OR gate will generate all sum bits.

So, the first block here will be something like these p and g generators. What are the inputs for p and g generators. We have two 4-bit numbers; a_0 a_1 a_2 a_3, b_0 b_1 b_2 b_3 and then, we have another c_0 as another input.

So, using this, what we are going to generate? We are going to generate all p’s and g’s; p_0 p_1 p_2 p_3, g_0 g_1 g_2 g_3; then, this we are going to connect to carry generation network. So, this will generate all the carries c_1 c_2 c_3 c_4, then we have final sum generation. So, this will final outputs sum s_0, s_1, s_2, s_3 and final carry c_4. So, these are overall inputs and these are overall outputs.

In between, we are using some variables, this you have to define as a wire. As your discussed in the earlier classes also, these are not accessible, so we have to define these as a wire. So, how many wires are required total? So, we require four for p's, four for g, four for carries.

So, wire [3 : 0] p , g; wire [4 : 1] c; this is p_3 to p_0, g_3 to g_0, c_4 to c_1, c_0 is actually input carry; this is about the initialization. So, what are the different key words required to generate the first step? This is step 1, step 2, step 3. So, for step 1, so what is there inside the circuitry? We know that this p's and g's will be generated using AND gates and exclusive OR gates.

So, I will write assign; first I will generate all p's. So what is the expression for p_0? p_0 is exclusive OR operations, g_0 is AND operations. Here you can write instead of this suffix 0, you can write p 0 is equal to a_0 exclusive OR b_0, I am going to use only single assign for all the p's; p_1 is equal to a_1, we can write in the bracket also. It is up to you, a_1 exclusive OR b_1 , p_2 is equal to a_2 exclusive OR b_2 , p_3 is equal to a_3 exclusive OR b_3, then semicolon.

So, p's are over; similarly, you have to write assign g's, g_0 is AND operation a_0 AND with b_0 , I am going to use only single assign for this and operation also g_1 is equal to a_1 AND with b_1 , g_2, g_2 is equal to a_2 AND with b_2 , g_3 is equal to a_3 AND with b_3 semicolon. So, second assign is for all g's. So, this will give the step 1 and in the step 2, you have to use carry generation circuit.

Assign c_1 is equal to what is the expression for c_1? c_1 is $g_0 + p_0 c_0$; $g_0 +$ is OR operation is this p_0 AND c_0. Here, also I can use single assign using commas. Similarly, you have to write c_2 is equal to basically you have to write this Boolean expression; then for c_3, this Boolean expression; for c_4, this Boolean expression.

So, we have only AND operation, OR operations only. So, AND operation with this, OR operation with this. So, I am not writing c_2. So, this is a Boolean expression. We have comma c_3, this is another Boolean expression; comma c_4, is another Boolean expression using AND and OR gates semicolon. So, this is second step. In third step is to generate s_0, s_1, s_2, s_3 assign.

This c_4 is generated here, s_0 is equal to this is exclusive OR between p_0 and c_0 right; p $_0$ exclusive OR within a_0, single assign s_1 is equal to p_1 exclusive OR with a_1, $s_{2,}$ p_2 exclusive OR with $a_{2,}$ $s_{3,}$ p_3 exclusive OR with a_3. Then, this is end module.

So, if you want to write this dataflow modelling, we have a simple code; whereas, if you want to use this gate level modelling, it becomes very difficult because we have lot of gates here. So, this is about the VERILOG code using dataflow modelling for a Carry Lookahead Adder.

Thank you.