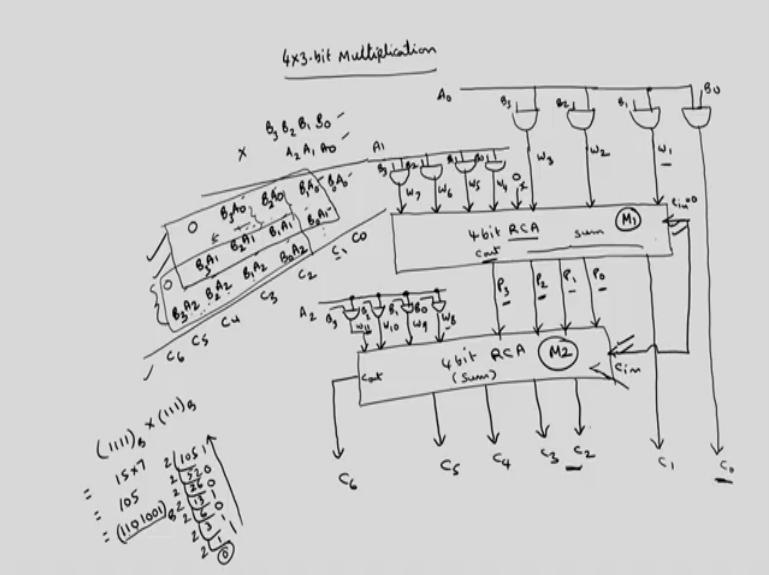
System Design Through VERILOG
Prof. Shaik Rafi Ahmed
Department of Electrical and Electronics Engineering
Indian Institute of Technology, Guwahati

Gate level modeling - I
Lecture - 06
Multiplier and comparator

Ok. In the last class, we have discussed about this 2 by 2 multiplier. So, today we will discuss about the 4-bit by 3-bit multiplier and we are going to show that this can be implemented by using a 4-bit ripple carry adder. So, while writing the Verilog code you can instantiate the 4-bit ripple carry adder code, ok.

(Refer Slide Time: 00:52)



So, if you consider the 4-bit by 3-bit multiplier, basically I am taking this 4-bit number as B 3, B 2, B 1, B 0 into A 2, A 1, A 0. So, if you perform this multiplication, so A 0 B 0, B 1 A 0, B 2 A 0, B 3 A 0. And if you multiply with A 1, B 0 A 1, B 1 A 1, B 2 A 1 and B 3 A 1. If you multiply with A 2, B 0 A 2, B 1 A 2, B 2 A 2, and B 3 A 2. So, by multiplying this 4-bit by 3-bit number the number of resultant output bits will be; so, this B 0 A 0 will be directly C0, by adding these 2-bits we will get C 1.

So, these 3-bits C 2, C 3, C 4, C 5 here we may get carry from this stage, so by adding these two we may get one carry also. So, totally we require 7 bits. This you can visually verify using this the maximum numbers of these 4-bit and 3-bit. So, the maximum 4-bit number is 1111 in binary into 111 is the 3-bit maximum number in binary. So, this is in decimal this is 15 into 7. So, this is equivalent to 105.

So, to represent this 105 we require the 7 bits, ok. 105 if you divide successively with 2, so this is 52, 1 is the reminder, if 26, 0 is the reminder; 13, 0 is the reminder; 6, 1 is the reminder; 3, 0 is the reminder; 1, 1 is the reminder; 0, 1 is the reminder, till this is 0 we have to continue. The value of this 105 in binary will be you have to read from bottom to top 1101001 in binary. So, how many bits are there? 7 bits. So, 7 bits are required, ok.

So, now how to implement this using binary adder? So, basically you have to first generate all these product terms. How many product terms are there? This is 12 products terms are required. So, we can use 12 AND gate produce these 12 product terms. So, basically I am giving first A0 as a line with 4 AND gates which is common to 4 AND gates.

This is A 0 line. So, you can give this as a input for one AND gate whose other input is B 0, you can give this input for the another AND gate whose second bit is B 1, this has one bit for the other AND gate whose other bit is B 2, then we have fourth AND gate whose other bit is B 3. So, this directly A 0 B 0 is your C 0, this will come as a final output C 0.

Now, to get C 1 you have to add B 1 A 0 with B 0 A 1. So, you have to give this as input for this another A 1 line. Here also we have 4 AND gates. So, this bit is B 3, B 2, B 1, B0. So, you have to basically add these 3-bits along with 0 and these 4-bits. Here basically I am going to perform. So, by inserting a 0 here. So, you can insert as many 0s as you want at the MSB position. I want to perform basically this 4-bit addition.

So, in order to perform this 4-bit addition, we can use 4-bit parallel adder such as ripple carry adder. So, I am going to use here ripple carry adder. In fact, at the first adder is carry look ahead adder, actually I have discussed about ripple carry adder that is why I am writing, otherwise you can use the look ahead carry adder also. So, the 4th bit of the first set of the

number is 0, ok. So, this addition will be performed by this 4-bit ripple carry adder. So, directly the last bit will be your C 1 bit.

So, here this the sum bit of this one is C 1 bit and the carries will be propagated inside this because this is the ripple carry adder. So, the carry from this addition will be propagated to this, this addition will be propagated to this and so on, that will take place inside. And the remaining 3-bits total you will get 5 bits, ok. You will get remaining 4-bits along with the carry.

So, total this is sum bits and this is carry out. Total 5 bits will be available in that, one bit will be directly taken as the output, remaining 4-bits will be given to then here I can insert a 0, then I will perform addition of these 4-bits from here to here using another ripple carry adder, this, ok. For that this is one set of the numbers, 4-bit ripple carry adder.

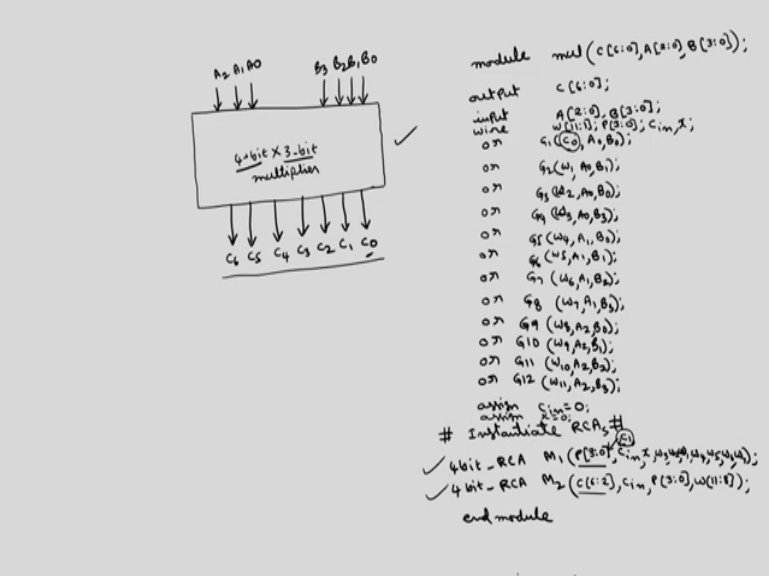
The second set of the numbers will be B 3 A 2, B 2 A 2, B 1 A 2, B 0 A 2, this can be generated through, so this line is A 1 line, this is A 2 line. So, along with this A 2 you have to generate 4 signals through 4 AND gates. This is one AND gate, second AND gate, third AND gate, fourth AND gate. So, this bit is common, the other bits are here, this bit is B 3, this is B 2, this is B 1, and this is B 0.

Then, we will get finally, this carry out total 5 bits we will get, here the 4-bits, sum bits, this is carry out, this is sum bits. So, these sum bits will be C 2, C 3, C 4, C 5, C 6. This is the final implementation of a 4-bit by 3bit multiplier using two 4-bit ripple carry adders along with 12 AND gates, 2 input AND gates.

Now, because we have already written the code correspond to this 4-bit ripple carry adder you have to instantiate that code. So, basically you call these outputs of this AND gates as wires.

So, this is basically C 0 is directly the output, ok. No need of wire here. So, the remaining we require the wires. So, we can call these wires as W 1, W 2, W 3, W 4, W 5, W 6, W 7, W 8, W 9, W 10, W 11, this wire you have to define, and in addition to that these also you have define as a wire, ok. Let us call this one as P 3, P 2, P 1, P0.

(Refer Slide Time: 11:00)



So, now, what will be the Verilog code correspond to this one? So, if I take this whole block a 4-bit by 3-bit multiplier, if I consider as a whole block, what are the inputs and outputs?

We have one 4-bit number input B 3, B 2, B 1, B 0, this is block diagram. Inside this we will get that circuit we have discussed in the earlier slide. This is A 2, A 1, A0, one 4-bit number, another 3-bit number. Then we have 7 outputs I think, so total is 7 outputs C 0, C 1, C 2, C 3, C 4, C 5, C 6. This is the block diagram of 4-bit by 3-bit multiplier.

So, we can define this as a module, I will give the name as mul for multiplier. So, what are the inputs and outputs of this multiplier? So, we have the outputs C 6 : 0, then inputs are A 2 : 0, B 3 :0. Then, we have we can instantiate or before that will generate all the wires, ok.

So, what are the wires? Wire 1 is directly C 0 is the output, we will get C 0 output through the AND gate. So, we will get C 0. And after that the wires are this W 1, W 2s goes up to W 11, we can generate a through this A 0 B 1, A 0 B 2, A 0 B 3, similarly, A 1 B 0, A 1 B 1, so on, ok.

So, here of course, this carry in has to be 0, the same carry in has to be applied to this also, this carry in, this carry in is equal to 0. So, I will define first end operations I will get directly

C 0 through this one or output of the OR gate is C 0 which is one of the output of the final 4-bit by 3-bit multiplier. The inputs are A 0 B 0.

So, totally we have W 11 comma 1, and then we require one more wire which is P 3 comma 0. These are the 2 wires required, right. So, these wires are required. So, I have got C 0 output, then I will generate the remaining all wired signals or C 1 is nothing but A 0 with B 1, C 2 is nothing but A 0 with B 2 or C 3 is nothing but this is W 1, W 2, W 3.

This is W 1, W 2, W 3, A 0 comma B 3, because these are the wires, this is W 1. This is C 0 is the first output, the remaining all are wires. So, you have not obtained the outputs, we have to obtain through the 4-bit ripple carry adders, this is W 1.

Then, W 1, W 2, W 3, then we have similarly W 4 is A 1 with B 0, W 5 is A 1 with B 1. We can give this as a name G 1, G 2, G 3, G 4, G 5, G 6, G 7, W 6 comma A 1 comma B 2 OR G 8, W 7 comma A 1 comma B 3, OR G 9, W 8 comma A 2 comma B 0, OR G 10, W 9 A 2 comma B 1, OR G 11, W 10 A 2 comma B 2, OR G 12, W 11 comma A 2 comma B 3. This will generate all the wires, so up to here, ok.

Now, to get P 3, P 2, P 1, P 0, you have to instantiate this 4-bit ripple carry adder. To get the final result you have to instantiate this, ok with the C in is equal to 0, ok. So, we can assign C in is equal to 0, we can take this as input. Even you can define this wire for this C in also, otherwise because this is not external to this block. So, I am defining this wire as C in. So, I will assign C in is equal to 0, then you have to instantiate ripple carry adders. I will give this module as M 1 similar to the previous example, this I will call as M 2.

For M 1 what are the inputs? So, the name that you have given for the code of this RCAs is 4-bit_RCA M 1. So, what are the outputs and inputs of these? Outputs are P 3, P 2, P 1, P 0 and you will get of course some here C out sorry, including the C out only this is P 3, P 2, P 1, P0 because this is C out, ok, so P 3, P 2, P 1, P0.

So, outputs are P 3 comma 0, the inputs are C in is the one input, the other inputs are this is 0, we can call this one has some x signal say, x also we will gave because this is internal I will define x also wire and I will assign x is equal to 0. Then x W 3, W 2, W 1 is one set of the numbers, another set of the number is W 7 to W 4. x, I will define x as a wire here. Assign x

is equal to 0. So, x is one input, C in is another input, the other inputs are x comma W 3, W 2, W 1, W0, then we have W 4, W 5, W 6, W 7. So, this will produce P 3, P 2, P 1, P 0, ok.

Then again you have to instantiate this 4-bit ripple carry adder whose name I have given as M 2. Now, what are the outputs and inputs of this? So, the inputs are C in is there again, same C in, and then P 3, P 2, P 1, P0 is one set of the input, W 11, W 10, W 9, W 8, are the other inputs, then the final outputs are C 6 to C 2. So, for this ripple carry adder C 1 is also output, you can write C 1 also as the output. Here you have to write C 1 also because C 1 is also output, ok.

Then, the output final outputs are C 6, 2. C 1, C 0 already generated, so remaining all are C 2. These are the outputs, the inputs are C in is one input, then P 3 comma 0 are the other inputs, this P 3 comma 0, then C in and we have W 8 to 11. So, this will produce the C 6, C 5, C 4, C 3, C 2. C 1 has been produced in the previous 4-bit ripple carry adder, C 0 through this AND gate, so now, we have got all the outputs. So, this is end module.

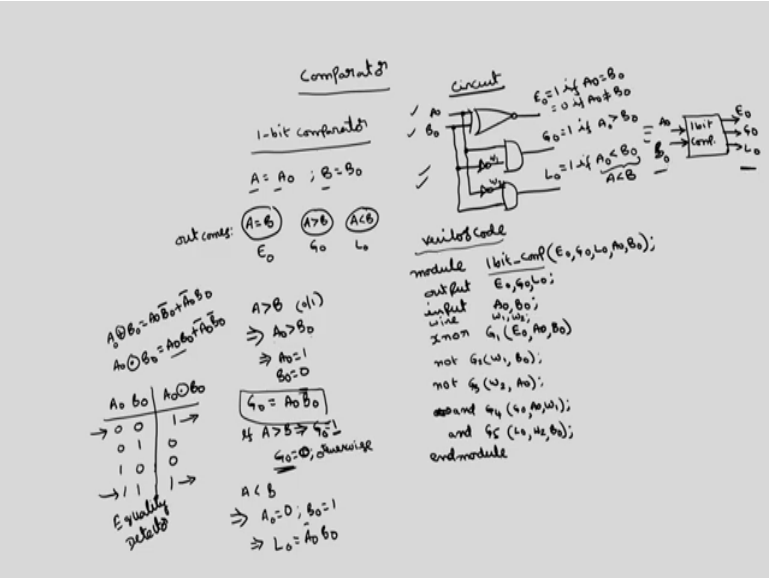
Here you have got C 1 output, here C 2 to C 6, and here you have got C 0. So, total this C 6 to C 0 will be generated. This is how you can use same 4-bit ripple carry adder, even to implement a multiplier also. So, we have discussed a 4-bit by 3-bit multiplier in a similar fashion. We can design any multiplier, in general N bit by M bit multiplier, N and M can be any values, ok. So, we can implement this by using basically ripple carry adders.

So, in fact, this the multiplier that I have discussed is unsigned multiplier. I have not considered the signs. But in general, in microprocessor and the DSP processors, normally we will use a signed multiplier. So, signed multipliers we may discuss while discussing about the behavioral modeling, ok.

And the next arithmetic block is we have discussed about the adder subtractor, we can combine in a single block, then we have discussed about the multiplier. The next, I mean arithmetic operation is division, but division is not a combinational circuit, we are discussing about the combinational circuit. So, this division also actually we will discuss while discussing about the behavioral modeling because division circuit is somewhat complex, ok.

So, with these arithmetic operations; another important arithmetic operation is the comparison.

(Refer Slide Time: 23:06)



So, the circuit which perform the comparison operation is called comparator. So, how to design a comparator? So, I will start with a single bit comparator and then I will go to the higher order comparators by instantiating the lower order comparators.

So, if I take 1-bit comparator there are two numbers A and B, only single bit, this is A 0, this is B 0, this is LSB bit and MSB bit itself, only 1-bit. So, a comparator is a logical circuit which compares the two numbers and the outcome of this comparator is 3 outputs.

One is equal, another is greater, the third one is less. So, if I compare this A and B, so what are the possibilities? A can be equal to B, I will call this one as E 0 because I am using only 0. A can be greater than B, I will call this one as G 0. A can be less than B this is L 0. So, these are the outcomes of the comparison operator, outcomes of a comparator operation.

Now, what will be the circuit diagram for this E 0, G 0, L 0? How do you realize this signal? Ok. For that you have this is 1-bit logic means a simple logic. So, we know that exclusive NOR gate, this is the circuit diagram, exclusive NOR gate will act as an equality detector. This is exclusive NOR gate this will acts as a equality detector. If I give A 0, B 0 as the input

we will get output as E 0. E 0 is equal to 1, if A 0 is equal to B 0 otherwise 0. This is equal to 0, if A 0 is not equal to B 0.

So, exclusive OR is nothing, but A 0 exclusive NOR with B 0, this is the exclusive NOR operation. A exclusive OR B is this is the operation. This is A 0 B 0 bar plus A 0 bar B 0. Whereas, this is A 0 B 0 plus A 0 bar B 0 bar.

So, if I take the truth table A 0 B 0 and then A 0 exclusive NOR with B 0. 0, 0, this is 0 0 1 1 1, 0 1, 0, 1 0 also 0, 1 1 is this will become 1, so 1. So, if the input bits are equal, output is 1 otherwise output is 0. So, this exclusive NOR gate acts as an equality detector. So, basically exclusive NOR gate will be used for checking the equality.

Now, how to check A greater than B? So, A greater than B implies only 1-bit is there, so A 0 should be greater than B 0. So, this A 0 and B 0 are can be 0 or 1. So, A0 is greater than B 0 mean implies, A 0 is equal to 1, B 0 is equal to 0. So, to get G 0 greater than equal to 1, this is equal to A 0 B 0 bar, ok. So, if A is greater than B implies you will get G 0 equal to 1, and G 0 is equal to 0 otherwise. Whether this is equal or less than G 0 is 0, if A is greater than B, G is 1.

So, how to implement this? Only 1-bit this is basically the expression for the greater than operation is A 0 B 0 bar. So, we take A 0 as it is and B 0 through NOT gate, you apply to the AND gate this will give G 0. So, G 0 is equal to 1, if A is A 0 is greater than B 0, A 0 itself is A here because single bit.

Similarly, you can obtain this less than as A is less than B implies A 0 is 0, B 0 is equal to 1, so implies less than operation if I write L 0 is nothing but A 0 bar B 0. This is just compliment of that of this greater than. So, this you have to take with complement, this you have to take without complement. This is less than, L 0 is equal to 1 if A 0 is less than B 0, that implies A is less than B because only single bit number. So, you see the basic circuit diagram of a 1-bit comparator. So, we have 2-bits as the inputs and 3 outcomes equal, greater than or less than.

So, how to write the code here? Well it is a very simple. Verilog code for this one is, so you can define module 1-bit_comparison, I will give the name as 1-bit_comparator. The outputs

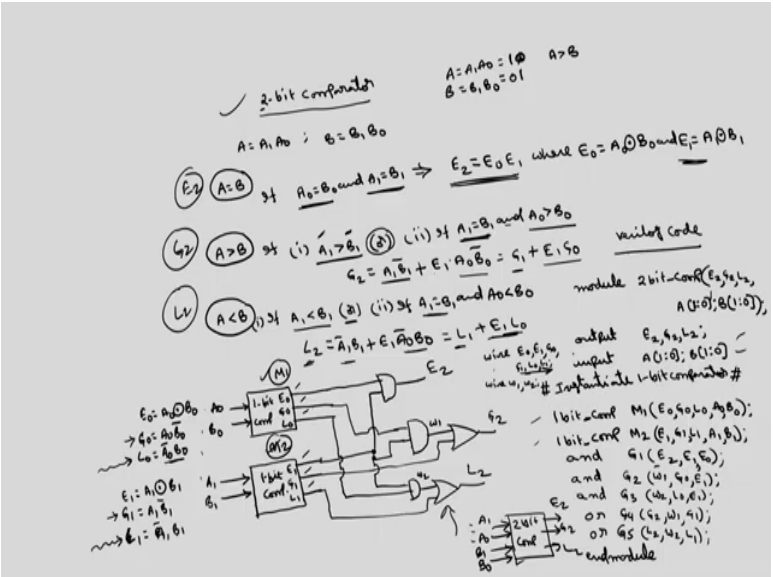
are E 0, G 0, L 0 inputs are A 0, B 0. Output E 0, G 0, L 0, input A 0, B 0, XNOR we give name as gate 1, output is E 0, input is A 0, B 0. And you have to define 2 wires also, you call this one as W 1, W 2, wire W 1 comma W 2, NOT W 1, the input is B 0, NOT wire is W 2, input is A 0.

Now, using these wires, so the greater than operation will be AND, give this as G 2, G 3, G 4, the output of the AND gate is G 0, the inputs are W 1 and A 0, AND G 5 output is L 0, input is W 2 and B 0, end module. This is the basic Verilog code correspond to 1-bit comparator.

Now, actually in order to synthesize the circuit in an easier manner, so you have to construct the bigger circuits using smaller circuits. So, if I take this as a block diagram, this is equivalent to, so the inputs are A 0, B 0, outputs are E 0, G 0, L 0, this is 1-bit comparator. If I can construct the 2-bit comparator using 2 single bit comparators I can instantiate this program. So, I have given the name as 1-bit comparator. So, the inputs are A 0, B 0, outputs are E 0, G 0, L 0.

So, how to realize a 2-bit comparator? So, we will proceed in a progressive manner to the higher order comparators, 2-bit comparator.

(Refer Slide Time: 32:00)



So, A is a 2-bit number, A 1, A 0; A 1 is MSB, A 0 is LSB. B is B 1, B0, B 1 is MSB, B 0 is LSB. Now, to check A is equal to B. So, what are the condition for A is equal to B, A is equal

to B if A 0 is equal to B 0 and A 1 is equal to B 1, then only we can say that A is equal to B. So, we have 2-bits, both the bits have to be equal.

So, how to check this equality? We have called this one as that implies, so the expression for the equality I will call as E 2, because this is 2-bit comparator I will call as E 2. E 2 is equal to, to check this equality this is E 0, to check this equality E 1, where what is E 0 and E 1? E0 is exclusive NOR between A 0 and B 0 and E 1 is exclusive NOR between A 1 and B 1. So, basically E 2 is E 0 into E 1.

Then A less than B or A greater than B, it is up to you. A greater than B, how to check A greater than B? We have 2 numbers, We have 2 conditions, one is if A 1 is greater than B 1, regardless of A 0 and B 0 it is sufficient to say that A is greater than B, ok because this MSB bit. So, A is a 2-bit number A 1 A 0, B is a 2-bit number B 1 B 0. If I assume that this is 1, this is a do not care this can be 0 or 1. So, A 1 is greater than B 1 means A 1 is 1, B 1 is 0, this is 0, this is E 1, this is 1, if this is E 1 is 0 the least value. So, in any case A is greater than B.

So, if MSB bit of A is greater than the MSB bit of B, it is sufficient to say that A is greater than B. This is one condition. What is the second condition? If A 1 is equal to B 1 say then and if A 1 is equal to B 1, then A 0 should be greater than B 0.

These are the only 2 conditions under which we can say that A is greater than B. So, what is the Boolean expression for this? If I call this as G 2, this is equal to what is this? This is G 1. This is G 1 expression or if you want you can write in a Boolean expression A 1 greater than B 1 means A 1 B 1 bar.

This is nothing but E 1 this is OR, plus this is E 1 and this is AND, and A 0 greater than B0 means A 0 B 0 bar. So, this is the expression for G 2.

Similarly, the third condition A less than B, if under 2 conditions similar to this, if A 1 is less than B 1 it is sufficient to say that A is less than B or second condition is if A 1 is equal to B 1 and A 0 is less than B 0. These are the 2 conditions under which we can say that A is less than B.

So, what is the condition for this output L 2 if I call as? This is equal to A 1 less than B 1 means A 1 bar B 1 this is plus. So, A 1 B 1 means this is E 1, I have defined what is E 1 here and A 0 bar B 0. So, these are the expressions, ok. For 1-bit comparator what are these? For 1-bit comparator we have taken A 0 B 0 as this one, A 0 B 0, and you have got 3 expressions E 0, G 0, L 0. This is 1-bit comparator.

So, what is the expression for E 0? You have obtained is A 0 exclusive NOR between B 0, and G 0 is A 0 B 0 bar, L 0 is A 0 bar B 0.

Similarly, I will take another 1-bit comparator and I will give the inputs as A 1, B 1 and I will call the outputs as E 1, G 1, L 1. So, what are the expression for E 1? Is A 1 exclusive NOR with B 1, G 1 is nothing but A 1 B 1 bar, L l is equal to A 1 bar B 1. Now, we have got this expression. Now, how to implement? This overall E. So, this E 0, E 1 AND operation is nothing, but E. So, this E 2, we will call as E 2, output of this 2-bit comparator we are calling as this as E 2, this as G 2, this as L 2

So, what is the expression for E 2? E 2 is nothing, but E 0 E 1. You just I mean AND these 2 operations you will get E 2, this is E 2 is the output of these 2-bit comparator. Now, how to get G 2? G 2 is nothing, but A 1 B 1 bar, A 1 B 1 bar is nothing, but G 1. So, this you can write as G 1 plus E 1 as it is. And what is A 0 B 0 bar sorry this A 0 B 0 bar? Is G 0.

So, the expression for G 2 is equal to G 1 plus E 1 G0, ok. So, G 1 plus E 1, E 1 G 0, E 1 G 0. So, this is G 0 E 1 plus G 1 if we OR with this one we will get G 2. Similarly, to get the expression for L 2, A 1 bar B 1, A 1 bar B 1 is this which is L 1, this is equal to L 1 plus E 1. What is A 0 bar B 0? A 0bar B 0 is L 0, so that E 1 into L 0, ok. So, we can get this L 2 as E 1 L 0, L 0 AND E 1, these two you have to AND, then you have to OR with L 1, this is your L 2.

So, by instantiating this one-bit comparator twice using some additional gates we can implement a 2-bit comparator. So, what will be the code for this 2-bit comparator? Module, I will call as 2-bit comparator, 1-bit comparator I have given as one-bit cmp. So, what are the outputs and inputs of this 2-bit comparator? We have 4 inputs and 3 outputs, outputs are E 2, G 2, L 2 comma inputs are A 1 : 0, B 1 : 0.

Then, first you have to instantiate this lower order modules. Of course, we require some we can define output and input. Outputs are E 2, G 2, L 2, inputs A 1 : 0, B 1 : 0 because if I write this as a block, these are 4 inputs and 3 outputs, this is 2-bit comparator, A 1 A 0, B 1 B 0, this is E 2, G 2, L 2. So, these are the inputs outputs I have defined.

Now, we have to instantiate 1-bit comparators. So, the name that I have given is 1-bit_comp. We call this was module one M 1, you call this as module M 2, M 1. What are the inputs and outputs of the module 1?

Outputs are E 0, G 0, L 0, inputs are A 0, B 0. So, this will perform this and this will give outputs as E 0, L 0, G 0. Then you have to write another 1-bit comparator M 2, the outputs are E 1, G 1, L 1 , A 1 , B 1. Of course, you have to define this E 0, G 0, L 0, E 1, G 1, L 1 as a wire, ok. You can write somewhere here wire, E 0 , E 1 , G 0 comma G 1 , L 0 , L 1 as a wire, ok.

Then, these two 1-bit comparators we have instantiated then final result to get E is 2. So, what is the expression for E 2? E 2 is nothing, but E 1 E 0, ok. You have taken AND gate whose you call this as G 1 gate 1, output is E 2, inputs are E 1 and E 0 which you have generated using these two 1-bit comparators, we have defined these as wires, ok. So, E 2 has obtained.

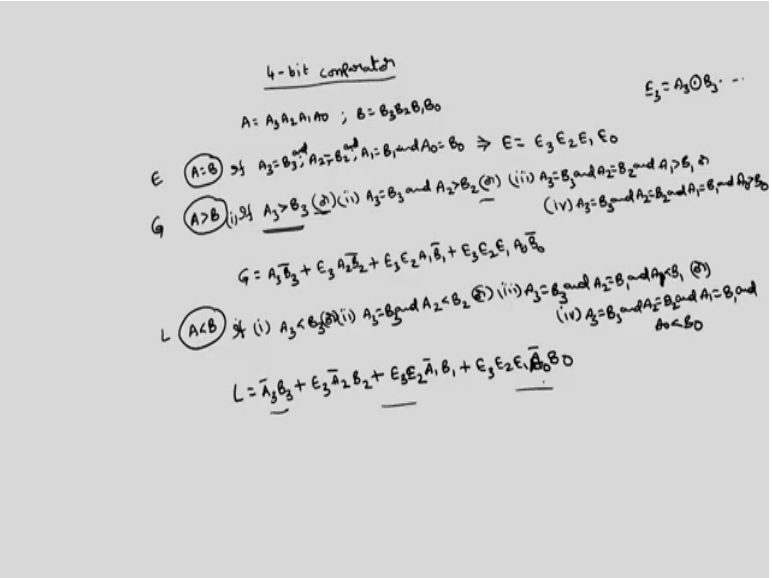
To obtain G 2, G 1 plus E 1 G 0. So, first you obtain E 1 G 0 you call this one as some other wire W 1, this you call as W 2, so we can define here W 1, W 2 also wires. So, you perform and operation G 2, output is W 1, inputs are this is G 0 and E 1. Similarly, AND G 3 W 2, this L 0 , E 1.

Now, once if you obtain this W 1, W 2, so G 2 is nothing but G 1 with W 1, L 2 is nothing but L 1 with W 2, OR gate G 4, G 2 is the output which is greater than output the inputs are W 1 and G 1, OR G 5, L 2 is the output to get the L 2 output this you have defined as W 2, the other one is L 1.

This L 1 you have defined as a wire which you have generated using this 1-bit comparators. So, we got the final three outputs greater than, less than, and then equal E 2. This is end

module. This is how we can construct the larger comparators knowing the lower comparator codes, ok.

(Refer Slide Time: 46:34)



Now, I will extent to the 4-bit comparator I will just explain the theory. So, you can write in a similar manner the Verilog code. So, 4-bit comparator, this you can construct using two 2-bit comparators. So, basically A is having 4-bits A 3, A 2, A 1, A0; B is having 4-bit numbers B 3, B 2, B 1, B 0. Here A 0, B 0 are LSBs, and A 3 and B 3 are MSBs.

So, A is equal to B, how to check this? If A 3 is equal to B 3, A 2 is equal to B 2, A 1 is equal to B 1, this is all AND, AND A 0is equal to B 0. So, this we are calling A 1 A 0 as this implies, if I call this as E finally; and if I call this one as E 3, A 2 is equal to B 2 if I call as E 2, E 1, E 0, ok, where E 3 is equal to A 3 exclusive NOR with B 3 and so on.

A greater than B, if I call this as E, this as G finally. So, if what is the condition? There are 2-3 conditions. One is if A 3 is greater than B 3, it is enough to say that A is greater than B. If otherwise or, what is the condition, second condition? If say MSB bit is equal and if A 2 is also greater than B 2 it is enough, or if first 2-bits are same, A 3 is equal to B 3 and A 2 is equal to B 2.

Then, the next bit has to be A 1 is greater than B 1; or the last term in condition is if the first 3-bits are same A 2 is equal to B 2, and A 1 is equal to B 1 and A 0 is greater than B 0, these are the conditions under which we can say that this A is greater than B.

So, what is the expression for G? So, you can directly write the expression. So, A 3 greater than B 3 means A 3 B 3 bar this, or, we have to just map this statement onto the Boolean expression, OR means plus. A 3 is equal to B 3 is E 3, A 2 greater than B 2 means A 2 B 2 bar OR A 3 is B 3 is E 2, A 2 B 2 is E 2 and A 1 B 1 bar plus E 3, E 2, E 1, E 0; E 3, E 2, E 1, and then A 0 B 0 bar. This is the expression for G, ok

Similarly, if I take the less than expression, A less than B. So, what are the conditions? If one condition is A 3 should be less than B 3 OR second condition is A 3 is equal to B 3 AND A 2 is less than B 2 OR third condition is A 3 is equal to B 3 AND A 2 is equal to B 2 AND A 0 is A 1 is less than B 1 OR forth condition is A 3 is equal to B 3 AND A 2 is equal to B 2 AND A 1 is equal to B 1 AND A 0 is less than B0. So, the expression for less will be A 3 bar B 3 OR is plus, E 3 A 2 bar B 2, OR E 3 E 2 A 1 bar B 1 plus E 3, E 2, E 1, E 0 sorry A 0 bar B 0, ok.

Now, we can construct these 4-bit comparator using 2-bit comparators by appropriately defining these values, then you can instantiate 2-bit comparator twice and with some external gates such as AND OR gates, we can construct this 4-bit comparator, ok.

So, see about this the next arithmetic circuit which is a 4-bit comparator. So, next we will discuss some of the control blocks such as decoders, encoders and all.

Thank you.