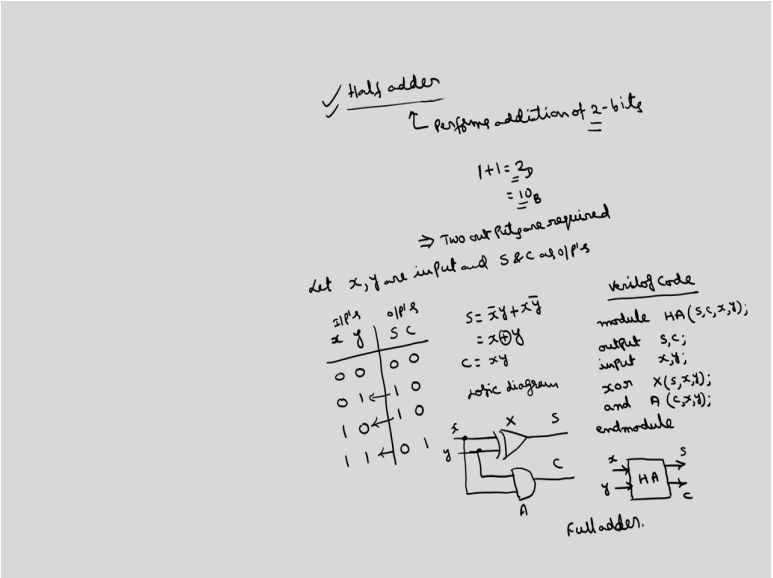
**System Design Through VERILOG**
**Prof. Shaik Rafi Ahmed**
**Department of Electrical and Electronics Engineering**
**Indian Institute of Technology, Guwahati**

**Gate level modeling - I**
**Lecture - 04**
**Half adder, full adder and ripple carry adder**

Ok in the last class, we are discussing about the Gate level modeling. So, in that, we started with a simple circuit and then, I have discussed about the design of the combinational logic and explained that design with an example of finding the square of a 3-bit number and finally, we have drawn the Verilog code for that one ok. So, today we will discuss about some of the commonly used combinational circuits. So, the first combinational circuit that I am going to discuss is half adder.

(Refer Slide Time: 01:09)




So, half adder is a combinational circuit which performs the addition of 2-bits, this will perform the addition of 2-bits. As we have discussed in the design procedure so, the very first step is so, we have to find out the number of inputs available and the number of outputs required. So, here, this is by definition the half adder will add 2-bits of the information.

Now, the question is how many number of bits are required to represent the outputs. So, that can be obtained by taking the maximum possible combination, 2-bits means the maximum possible combination is 1 plus 1 in binary or you can take in binary or decimal. So, if I take the decimal because 2-bits so, single bit can be either 0 or 1. So, if I take this in decimal this is 2. So, in order to represent this 2 in binary, how many bits are required? 2 is 10 binary.

This suffix 10 is not the binary number 10, but this is decimal. So, if you want to avoid the confusion because this binary equivalent of 2 is also 10 so, I will write this 1 as D, D stands for decimal and this 2 I will write as B, B stands for binary. So, the maximum possible output is 2 in decimal, the corresponding 2-bit binary equivalent is 10 so, two outputs are required, this will indicate that two outputs are required.

Next, we have to assign some letter symbols for the inputs and outputs. So, let us call the inputs as x, y or normally, we will call the outputs of the half adder as sum and carry. Then, you have to draw the truth table. For all combinations of the inputs, what are the outputs. Sum and carry, this is input and output. So, you take all the four combinations. So, 00 is sum is also 0 carry is also 0. 01 sum is 1 carry is 0. 10 also sum is 1 carry is 0. 11 is carry is 1 sum is 0.

So, the next step you have to write down the Boolean expressions for sum and carry. So, this is simple two variable truth table. So, we can easily write by observation. So, wherever 1 is there, corresponding inputs you have to read.

Here 1 is there, x is assigned as 0 so, you have to take with complement, y is assigned without 0, without complement plus in the next term, it is opposite xy bar, this is nothing, but x XOR y. Carry is only one min term is there, this correspond to the inputs both are uncomplemented so, we have to take xy.

So, finally, we have to draw the logic diagram. We have exclusive OR gate, this will give sum, these two are the inputs x, y, and the same input x, y you have to apply to the AND gate, this will give carry. This is the design of half adder, but how to write the Verilog code correspond to this one? So, there are two inputs, two outputs ok.

So, you have to start with module, you give some name to the module I will call as HA half adder, then inputs and outputs you have to define S comma C comma x comma y and I will define what are the outputs and inputs? Output S comma C, input x comma y. Do we require any wires? We do not require because these two are accessible, these are input port, S and C are output ports ok.

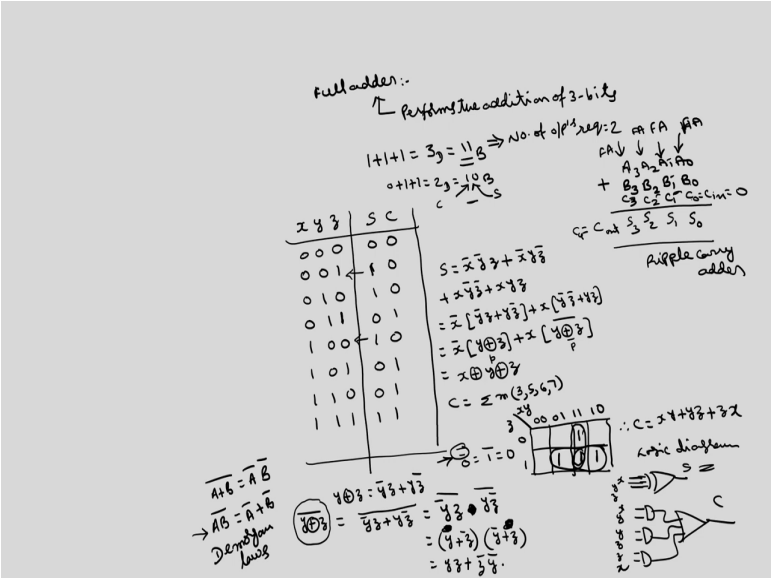
So, there is no internal connection which is not accessible to the input or output ports. So, no wires are required. So, simply sum is xor operation so, xor there is a primitive exclusive or if you want you can write the name, I will give this name as X, I will give this name as A say.

So, this name of the primitive is optional, its not compulsory, mandatory. So, what is the output of the XOR gate? Is s inputs are x and y. Similarly, and is the primitive of and gate A is the name that I have assigned, output is carry, inputs are same x and y, then end module.

So, one of the important features of this Verilog is we can instantiate these modules into the higher-level modules. So, if I call this as a lower-level module so, the next circuit is full adder ok, this is half adder. So, this half adder if I take as a block so, what are the inputs? x and y are the inputs, sum and carry are the outputs.

I can instantiate this half adder in the higher-level modules. So, what is the next higher-level module is full adder. You can construct the full adder using half adders. So, how to construct the full adder using half adder.

(Refer Slide Time: 07:53)




So, first of all we will discuss what is a full adder. So, if you want, I mean add the parallel numbers multi bit numbers say 4-bit addition, A 3, A 2, A 1, A 0 plus B 3, B 2, B 1, B 0. So, in order to perform this addition so, we have to do bit by bit, we have to first add A 0, B 0 ok.

So, to perform this A 0 plus B 0 because only 2-bits or half adder is enough, but in subsequent stages so, if I add these two bits as we have discussed in the earlier slide, you can get two outputs, sum as well as carry ok. So, this sum if I call as S 0, carry will be propagated here ok. Now, at the second stage, if you want to perform addition of 3-bits so, this is not possible with the half adder so, we have to go for the full adder.

So, full adder is a circuit which performs the addition of 3-bits. Now, again how many outputs are required for this particular full adder? Inputs are 3, it will add 3 bits. So, you can find out the number of outputs required in a similar way to that of half adder. The maximum possible binary bits all three are 1's, this is the worst case. So, this is in decimal 3. So, what will be this in binary? 11.

So, still there two outputs are required to represent these three. So, the number of outputs are required are 2. So, we will call again these outputs also sum and carry because if I add this A 1, B 1, C 1, I will get sum and carry so, that carry I will propagate to this one, sum I will

write here similarly, A 2, B 2, C 2 I will call sum as S 2 and C 3 as carry and this sum is S 3 and I will call this one as either C out or C 4, it is up to you, then also called as C 4.

Now, in order to have the uniformity, normally in case of binary parallel adders because this is half adder and these three are full adders to have the uniformity and if want to use all full adders, module d is normally we will make this C 0 which is equal to C in, we will make as 0 so that half adder will be replaced with full adder.

So, to perform the addition of 4-bits normally, we will use 4 full adders, this type of addition is called as ripple carry adder I will discuss this ripple carry adder after discussing about the full adder.

Now, coming to the full adder, again you take the same x, y, z as the inputs where z is the carry from the previous stage like here to here, here to here there is a carry that carry is z and we call the outputs as same sum and carry, S and C. So, this is 0, 0, 0 we have eight combinations ok so, sum is 0, carry is 0. 0, 0, 1 sum is 1, carry is 0. 0, 1, 0 same thing 1, 0. 0, 1, 1 is this is 1, 0, 1 plus 1 is nothing, but 0 plus; 0 plus 1 plus 1 is in decimal 2.

So, in binary is 1, 0. So, this first bit represents carry, and the second bit represents sum ok. So, this will be carry is 1, sum is 0. Similarly, 1, 0, 0 sum is 1, carry is 0. 1, 0, 1 you get 2 same similar to this ok. So, you will get 0, 1. 1, 1, 0 also wherever 2 bits are there, you will get the same result 0, 1. 1, 1, 1 both will be 1's ok.

So, if you write the expression for sum and carry so, this is actually there are four min terms, but that we cannot simplify using K-map because all single box combination even if we write this K-map simplification so, only thing is you can manipulate this. So, what is the expression here? This is correspond to x and y are 0 so, this is x bar y bar z because z is 1 plus the second term is x bar yz bar plus the third min term is xy bar z bar plus fourth one is xyz ok.

So, in the first two terms, x bar is common. If we take x bar as common, what is left? y bar z plus yz bar plus the next two terms, if I take x as common, y bar z bar plus yz. So, we know that this is x bar, and this is y exclusive OR z and this is x, this is y exclusive nor z. If I take

the complement of this one, you will get this, you can easily verify this. If you want you can do this also so, y exclusive OR z is y bar z plus yz bar.

So, if I take the complement of this, this is a whole complement y bar z plus yz bar so, this is equal to a kind of De Morgan's law that states that A plus B plus C sort up to whole bar is equal to A bar into B bar, this is one of the De Morgan's law which is on OR gate. Similarly, for AND gate, De Morgan's law is A bar plus B bar, these are called De Morgan's laws.

So, according to that this one will be if I assume y bar z as a single term, y bar z whole bar plus into yz bar whole bar. Again, if I apply the and De Morgan's theorem, this will be y double bar which is equal to y only plus z into this will be yz; z bar plus y bar plus z double bar this is nothing, but y itself. So, if I take this 0-bar; double bar, this is so 0 bar is 1, again 1 bar is 0 so, this double bar we can neglect.

So, this double bar we can neglect, this double bar we can neglect, y into y bar is 0, y into z this is equal to y into z plus z bar y bar, z, z bar is 0 so, this is nothing, but this exclusive nor operation. So, if I take the compliment of Exclusive OR gate, you will get this. So, now, this is again in the form of x bar into some p if I call this as p, this is p bar.

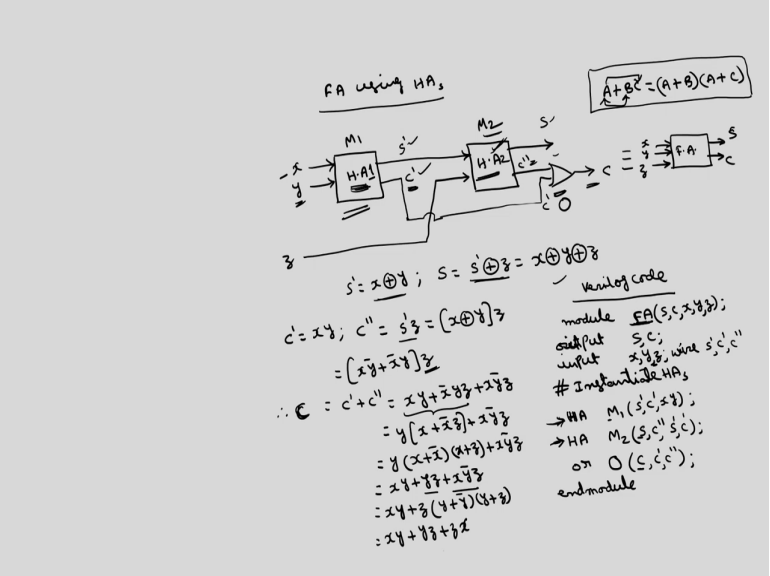
So, x exclusive or between p and x exclusive or so, p is nothing, but y, z. Similarly, if I take for the carry, we can simplify the carry using K-map, exclusive or operations we cannot solve ok. So, here what are these mean terms 3, 5, 6, 7. This I am taking xy, I will take the MSB as x, z as LSB you can take xy here or z here, you will get the final same result, but normally, I will follow this notation.

So, min terms are 3, 011 is 3. 101 is 5, 110 is 6, 111 is 7. So, these are three two box combinations, this is one, this is one and this is one. So, therefore, simplified expression for carry will be xy plus yz plus zx.

Now, we can draw the simply circuit diagram of this one and we can implement by using Verilog, this is one way, but what I want to do here is logic diagram of this one is this exclusive OR gate three inputs x, y, z, this will give sum and carry is you can take x y plus yz plus zx finally, we can add, this will give carry.

This ys and y are same only ok z, x. We can write the Verilog code correspond to this diagram, but I want to explain how to instantiate these smaller modules onto the larger modules. So, I will show the implementation of the full adder using the half adders because I have written the code for the half adder, I want to instantiate this half adder onto the full adder ok.

(Refer Slide Time: 18:02)




So, how to implement full adder using half adders. So, you need two half adders and one OR gate to implement a full adder. This is one half adder; this is another half adder. So, I will give two inputs to this half adder because half adder will be having two inputs only x, y. So, the sum of this I will give to another input of second half adder, the third input z I will give as input for the second half adder ok.

So, this will directly give the sum and the final carry, to get the final carry what you have to do is you take the carry of this one, carry of this one, you apply these two to OR gate. This is final carry of full adder. So, we can call this carry as C`, this carry you call as C``, this we call as s` this is the intermediate sum ok. So, we can easily verify that this will perform as a full adder because we know the expressions for the sum and carry. So, what is the expression for S dash is simply x exclusive or y.

So, what is the expression for s? S` exclusive or z, S` is xy so, if this is x exclusive or y exclusive or z, this is the direct relation. Whereas to get this C final expression for C so, what is C`? This C` is nothing, but ending between the inputs of that particular half adder so, C`is xy, then what is this C``? So, what are the inputs for this half adder? One is S ` into z, S` z are the input so, the sum is exclusive or between S` and z and the carry is AND operation ok.

So, this is S` is again x exclusive or y into z. So, this will be xy bar plus x bar y into z, this is equal to this is C` ok and C is nothing, but C`` and C` we have to or. So, this is C ` plus C ``, this is OR operation not plus ok, this is C, final C.

So, this is equal to what is C`? C ` is xy; C `is xy plus C`` is this x dash x bar yz plus xy bar z we can write in any way. So, this is equal to if we take y common in these two terms, what we will get? y into x plus x bar z plus xy bar z.

So, there is one important Boolean theorem which states that this is distributive law A plus BC, this is not valid in general algebra, but Boolean algebra, this is valid, A plus B into A plus C. So, A plus BC we have to OR with these two, we have to OR with these two, we have to take AND operation.

So, if I apply here this one, this is y into x plus x bar into x plus z plus xy bar z, x plus x bar is 1 ok. If x is 0, x bar becomes 1, 0 plus 1 or 1 plus 0 both are 1 only. So, this is simply again xy plus yz plus xy bar z. Again, from these two, you can take z as common, this is xy plus z again apply the same distributive law so, z if we take as common, what will be left? y plus y bar into y plus z.

So, this will be one again, we will get the final xy plus yz plus zx, this was the expression that we have obtained for the full adder using the K-map. So, we will get the same expression found implement by using two half adders ok. Now, if I want to write the Verilog code, you can instantiate half adder modules twice, I will call this 1 as H A1, this I will call as H A2 ok.

So, over all these full adders, we will be having how many inputs? So, you start with module so, you give the name as full adder and how many inputs and how many outputs if I write this as a block box, x, y, z are the inputs and sum and carry are outputs. So, I will write the outputs as S and C comma x, y, z semi colon so, I will give the explanation inputs are outputs

we can write otherwise first, output S comma C, input x y comma z. Now, I am going to instantiate these half adders twice.

This is the comment I am writing for better understanding so, instantiate half adders. So, for that so, whatever the name that we have given for the half adder in the earlier slide. So, the name I have given is HA ok so, we have to write HA only.

So, this name should be same as the module that I am going to write here, here we have to write the same name as that, we have written HA means we have to write HA only here HA, then we can give the name as 1 or 2, it is up to you some name you can give so, I will give otherwise to confuse this HA1, HA2, I will call this one as module 1 otherwise module 2.

So, I will write module 1. So, what are the inputs for this module and what are the outputs? For a half adder we know that two inputs and two outputs ok. So, we have to write so, we have to define this after this, we have to define there will be some wires are required because S`, C` are not accessible, we have to define wire, here I will write this will be just below this statement I will write here wire so, how many wires are required?

S`we have to define as a wire, C` we have to define as a wire, what else? C``this also we have to define as a wire ok. So, for M 1, what are the outputs and inputs? Outputs are S dash, y dash, C` this I have defined as a wire and the inputs are xy ok. So, this will perform all the internal operation that is taking place inside this.

So, what is the operation inside this half adder module, we have exclusive OR operation, AND operation that will be taken care by this instantiate operation so, no need to again write the primitives for exclusive OR gate and AND gate ok. So, that is one of the important features.

Then, I have to use another such half adder. So, I will write again the same code, this should be same, this should be same as the previous one because I have given the name for this half adder module as half adder. If we want to call this full adder, we have to use FA.

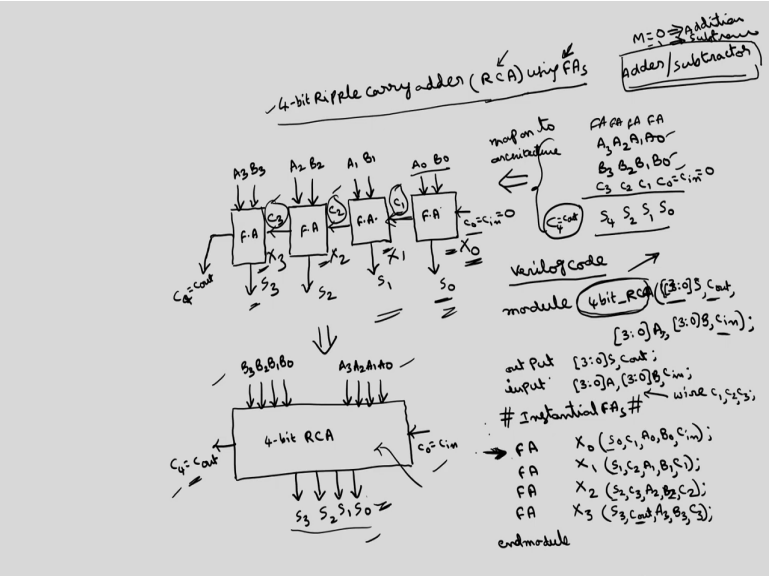
So, in the later I mean circuit, I am going to instantiate this again this entire full adder so, there I have to use the name FA and you have to give some option M 2 is the module I am

calling. So, for M 2 module, what are the inputs and output that you have to specify, no need to specify the internal circuitry of this ok.

So, what are the outputs? One is final sum s, another is C``which we have defined as a wire, the inputs are two wires, S`and C`. So, this will generate the outputs of half adders. Now, S is already generated to get the C, what you have to do? So, C is nothing, but OR so, through the OR gate you will get C. So, you can give the name for sum OR gate as say O, this was the name I have given for the OR gate.

So, what are the inputs and outputs? Output is C, inputs are C` and C``. So, this final carry is generated, a final sum is generated. So, this implementation of full adder is over. So, end module. So, this is how we can instantiate the lower-level modules to construct the higher-level modules ok. So, I have explained this with the full adder using half adders.

(Refer Slide Time: 28:05)




Now, I can use the full adder to construct some other big module so, which I will call as ripple carry adder. As I have defined this is my ripple carry adder, if I take 4-bit, 4 full adders are required ok. In general, for n-bit, n full adders are required. Here, I will discuss about 4-bit ripple carry adder.

As I have defined this 4-bits, I will call as A 3, A 2, A 1, A0 is the one number, B 3, B 2, B 1, B0 is another number. So, in order to perform this addition, as I have told four full adders are

required so, I will give this C 0 or I will call as Cin it is up to you as 0 because there will be no I mean the third bit, there is no carry from the previous stage if we start from here, then this will give sum as S 0, carry as C 1, this will give sum as S 1, carry as C 2, sum S 2, carry C 3, sum S 4 and this will give carry as C 4 or also called as C out as the carry.

So, how to implement this now? So, this is easier if I have the full adders. So, this RCA I am going to implement using full adders so that I can write the Verilog code for the RCA by instantiating the full adder codes by how? Written the code for the full adder using a by instantiating the half adders ok, this is how we can construct the larger modules.

So, first I will discuss how to I mean implement this 4-bit ripple carry adder using full adders. This is clear from this table, we have to map this onto the architecture ok, if we map this onto the architecture so, what will be the architecture?

So, as I told four full adders are required with inputs as A 0, B 0, C 0, A 1, B 1, C 1 so on up to A 3, B 3, C 3. So, I will start in the same manner I will go so, I will first write the LSB bit, this is one full adder. So, I will give the two inputs here, the carry here.

So, this C 0 or Cin I will make as 0, this I will give A 0, B 0. So, inside this full adder so, we have two half adders, inside the half adder, we have an exclusive OR gate and AND gate ok, this is abstract level representation.

So, this bit I will call as S 0 so, whatever the carry you have to propagate that is why the name ripple. So, you see like if you throw a stone onto the water so, there will be a ripples ok so, in a similar manner carry will be ripple through the subsequent stages that is why the name ripple carry adder.

Full adder A 1, B 1 and the previous carry is C 1. So, this will give S 0, S 1, then another full adder; another full adder finally, you will get C 5 which is equal to C out C 4 or C out, the inputs for this one are A 3, B 3 and this is C 3, this is A 2, B 2, C 2 so, this will give S 2, this will give S 3, this is the overall the implementation of the ripple carry adder using four full adders ok.

Now, how to write the Verilog code correspond to this? We can instantiate this full adder four times ok. So, overall, this is equivalent to a block I will call as a block at abstract level ok. So, what are the inputs and outputs of this block this I will call as 4-bit RCA?

Outputs are S 3, S 2, S 1, S 0, the input is C 0 or Cin, output is C 4 or C out, the other inputs are there are two sets of the inputs you have to give, this is one set of inputs, these are arrays as we have discussed this multiple bits can be represented in terms of array A 3, A 2, A 1, A 0 is one 4-bit array of one number, B 3, B 2, B 1, B0. So, this is the overall the representation of this 4-bit ripple carry adder.

So, you have to see only the overall inputs and outputs, internal connections, we do not bother here like C 1, C 2, C 3 so, these are all the internal connection, this we have to define as a wires, but in the final abstract model, we do not bother about what is C 1, C 2, C 3 and so on ok. So, what is the Verilog code here?

So, the module I will give the name as 4-bit under square RCA, this was the name I am giving for this, and I will define here this module as we can define some other module name ok. So, we can give here the name as sum X 0, X 1, this module names I am giving X 2, X 3, it is up to you, these are optional ok.

So, module 4-bit RCA, what are the overall outputs and inputs? Outputs are S 3 to S 0, this can be represented as 3 is to 0 comma S followed by S ok, this was the vector represented, this is 3. So, this will represent this S 3 to S 0. This is one output, what are the other output?

C out or C 4, I will call as it is up to you C out say, these are the outputs and what are the inputs? You have two sets of these A's and B's so, we can write in a similar manner 3:0 we can write both A, B also at the same time or we can write separately also 3 0 A comma 3: 0 B comma what are the other input? Cin ok.

Now, we can define the inputs, output separately. Output 3 : 0 S comma C out, input 3 : 0, we can write A, B at the same place also or we can write individually Cin. Now, I have defined this abstract level signals now, this, inside this I am going to instantiate full adder 4 times so, like we have instantiated half adders 2 times to write a code for its corresponding to FA so, I will instantiate full adders ok.

So, what is the name I have given for the full adder? So, you can refer to this full adder code, I have given name as FA so, we have to write the FA here also. So, whatever the name that we have to write here, we have to write this FA and I will call this first module as X 0. So, what are the inputs and outputs of X 0? So, you can see that the inputs are A 0, B 0, C 0 or Cin, output is S 0, C 1, you have to define this as a wire ok, here you define here as a wire.

What are the wires required? C 1 is not accessible C 2 and C 3. So, C 1 comma C 2 comma C 3. So, for X 0, the inputs are A 0, B 0, C in because I have defined C in so, I am not using C 0 and C 4 notation ok, and outputs are C 1 and S0. So, I will write here as S0, C 1 first outputs we have to write, the inputs are A 0, B 0, C in.

So, this will perform all the internal operations in full adder which is nothing, but 3-bit exclusive OR operation and to get the carry , AB, BC, AC operation and you have that again the full adder so, you have written this full adder using two half adders. So, this full adder again called two half adders ok. So, this is enough to represent this operation, no need of the internal details.

Again, the second full adder, the name, module name I have given as X 1. So, what are the inputs and outputs of this X 1? S 1 is one input, C 2 is another output, S 1 is output, C 1 is C 2 is output, the inputs are A 1, B 1 and C 1. Again, you have to instantiate two more times FA, this is X 2, for X 2 the output is S 2 comma C 3, inputs are A 2, B 2, C 2.

Then another time FA X 3, S 3 comma C out, I have called this as C out instead of C 4 because I have defined this as C out here ok. Then the inputs are A 3, B 3, then C. So, we have generated all S 0, S 1, S 2, S 3, these are the final output that we want ok and then, C out, C out also we have generated ok, so this is end module. This is one of the advantages of this Verilog code, we can instantiate the lower modules to construct the higher module, this is what is called hierarchical design ok.

Now, we will go a step advance. So, I want to again instantiate this 4-bit ripple carry adder to construct some other big circuit. So, what is that big circuit is so, we can have the module like adder / subtractor because in most of the processors, the ALU see if you want to design any processor, the one of the very important block in any processor is arithmetic logic unit.

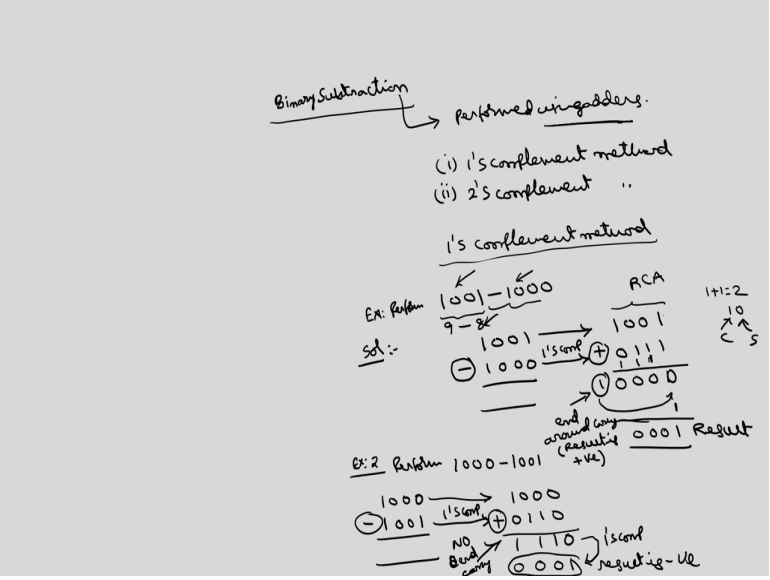
As the name implies, it performs various arithmetic operations such as addition, subtraction, multiplication, division, increment, decrement, comparison and various logical operations such as AND, OR, exclusive OR, shifting, rotating. So, we have to I mean design you know how to design these ALU. The first basic operations are addition, subtraction.

Now, again so, one important thing that has to be noted here is the subtraction. So, most of these subtraction circuit uses the same blocks of that of adder. So, no need to I mean design the subtractor circuit separately. So, we can share the resources of the adder and we can perform both the operation in single module ok. So, that type of circuit is called adder /subtractor.

So, in most of these ALU's, the adder and subtractor will not be separate circuits, but it will be in a single circuit so, some control signal is going to decide whether the operation is addition or subtraction, there will be some control signal like M, M is equal to 0, this same block will perform addition, if M is equal to 1, it will perform the subtraction. So, that type of modules will be there ok.

So, in order to develop that adder oblique subtractor, we need this several such 4-bit ripple carry adders ok. Of course, this ripple carry adder also there is some drawback like carry propagation, to get the final these carry and final sum so, it will take the propagation delay of four full adders ok to avoid this normally, we will use look ahead carry adder. So, I am not going into the details of look ahead carry adder ok.

(Refer Slide Time: 41:54)




So, using this ripple carry adders, I will generate the subtractor, binary subtraction. As I have told this binary subtraction will be performed by using the adders. So, how to perform this? There are two ways to perform this binary subtraction using adders so, one is called 1's complement method and the second one is 2's complement method. So, I will first discuss about the 1's complements method.

Suppose if I want to perform the subtraction as say 1001 minus 1000, this was the subtraction I want to perform. So, what we will do is we will take minuend subtrahend, I will take minuend as it is 1001 I will take as it is subtrahend, if you want to perform this subtraction 1000, this I will take as it is 1001 and I will take the 1's complement of the subtrahend.

So, you know how to get the 1's complement, you how to change the 0's to 1's and 1's to 0's. So, this will become now 0111, then you perform the addition. I have converted this subtraction into addition ok.

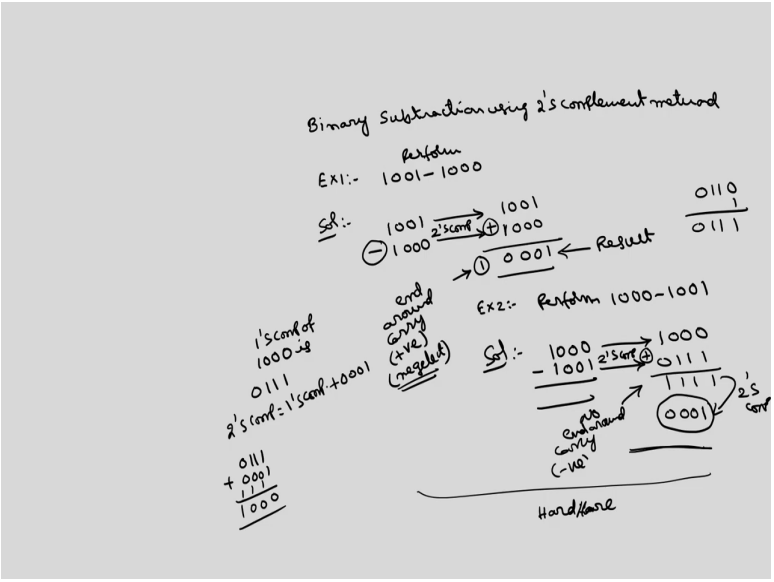
So, binary addition we know so, this can be performed by this 4-bit binary addition can be performed by using RCA that I have discussed in the earlier slide ok. So, this is 1 plus 1, 1 plus 1 is 1 is carry, sum is 0 because 1 plus 1 is 2, 2 can be represented as 10 so, sum is 0, carry is 1. So, 0 plus 1 plus 1 is 1 0, this is also 1 0, this is also 1 0, this is called end around carry.

So, if you get end around carry, this will represent that this is result is positive and how to get the result? We add this end around carry here so that this becomes 0001. So, this is the result and result is positive. There are two cases, this is one case where end around carry results ok, this is one so, it is clear that this is nothing, but 9 minus this is 8 so, 9 minus 8 is 1, this is the binary form of 1.

In example 2, I will take the reverse perform 8 minus 9 means 1000 minus 1001 ok. So, 1000 minus 1001 I have to perform, I will take this minuend directly, I will take the 1's complement of subtrahend so, which is 0110, then I will perform the addition, I will convert the subtraction into addition. So, this is 0111. So, there is no end around carry.

This will represent that no end around carry means result is negative and how to get the result? Result will be obtained by taking the 1's complement of the previous result. What is the 1's complement? This is 1000, this is the final result, but result is negative because there is no end around carry. This is how we can perform the subtraction using 1's complement method ok.

(Refer Slide Time: 46:24)




And subtraction in 2's complement method, I will directly explain with example. I will take the same example 9 minus 8 first 1001 minus 1000. So, 1001 minus 1000 I have to perform,

this I will take as it is, this I will take 2's complement because the method is 2's complement method.

So, what is the 2's complement of this? 1's complement of 1000 is 0111, 2's complement means 1's complement plus 1, so this is 0111 plus 0001, this is 8 so, this is 1000, then we have to perform addition I have converted the subtraction into addition 1000, this is 1 in end around carry.

So, this end around carry represents the result is positive and we have to neglect unlike 1's compliment where this end around carry has to be added whereas, here this has to be neglected ok. So, this itself is result. This is the why I mean most of the processors will use 2's complement type of the subtraction ok.

So, on the other hand, if there is no end around carry like if you want to perform 1000 minus 1001 so, 1000 minus 1001, this subtraction I am going to convert into addition by taking this minuend directly and 2's complement of the subtrahend, which is nothing, but so, 1's complement is 0110, 2's complement will be 0111, then I will perform the addition, this is 1111. So, there is no end around carry so, result is negative.

And how to get that result is by taking the 2's complement of again this. So, 1's complement is 0000, 2's complement will be 0001, this is the result, but most of the DSP processors and microprocessors uses 2's complement of our subtraction. So, I will implement this 2's complement of subtraction using the hardware, then I will write Verilog code correspond to this by instantiating, ripple carry adders four times ok. This we will discuss in the next lecture.

Thank you.