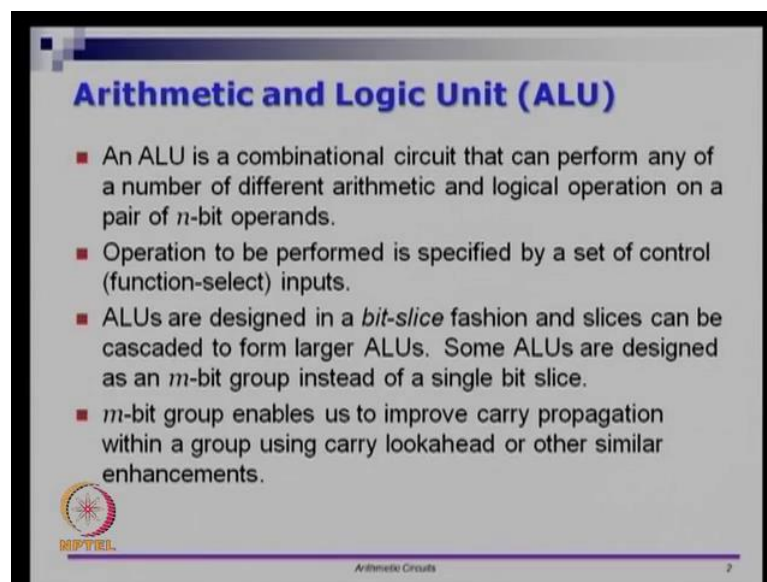


Digital Circuits and Systems
Prof. Shankar Balachandran
Department of Electrical Engineering
Indian Institute of Technology, Bombay
And
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module - 52
Multipliers and Other Circuits


We are in module 52; module 52 is about Multipliers and other arithmetic circuits, so we will start with multiplication. So, if you go and look at any processor, any processor typically has what is called as arithmetic and logic unit.

(Refer Slide Time: 00:34)



Arithmetic and Logic Unit (ALU)

- An ALU is a combinational circuit that can perform any of a number of different arithmetic and logical operation on a pair of n -bit operands.
- Operation to be performed is specified by a set of control (function-select) inputs.
- ALUs are designed in a *bit-slice* fashion and slices can be cascaded to form larger ALUs. Some ALUs are designed as an m -bit group instead of a single bit slice.
- m -bit group enables us to improve carry propagation within a group using carry lookahead or other similar enhancements.

 Arithmetic Circuits 2

An arithmetic and logic unit does all kinds of arithmetic calculations. So, arithmetic operations include addition, subtraction, multiplication, division and things like comparison and what not. And you may also have logical operations like logical AND, or bitwise AND, bitwise OR things like that are all done by ALU's. So, the operations to be performed are usually given by a control circuit.

So, based on the program that you have, you may have operations which say now add these numbers, subtract these two numbers and so on. Typically, you design them using what is called bit slice fashion. A bit slice fashion is for an ALU is one, in which you generate, you design the circuit for one stage and you just copy that for all the other

stages and that gives you the whole design. For example, when we did triple carry adder just the full adder is the slice, so that is a single slice and I put various slices together and I get the whole circuit.

So, think in terms of bread, when you cut bread, you get slices, but if when you put the slices together, you get a loaf. You can think that, it is a same analogy, when you put the bit slices together, you get the whole circuit and we looked at carry look ahead and ripple carry and so on, all of them have some kind of reputation. It is not that the circuit looks random, there was a very nice and repeatable structure that you will see in many of the circuits.

(Refer Slide Time: 02:04)

Unsigned Binary Multiplication

Multiplicand →

Multiplier →

Partial Product →

Product →

```

      1 0 1 0 1 0
    × 1 0 1 1
    -----
      1 0 1 0 1 0
     0 0 0 0 0 0
    1 0 1 0 1 0
   1 0 1 0 1 0
  -----
  1 1 1 0 0 1 1 0
          
```

} Partial Product Array

■ Consider unsigned m - and n -bit integers, X and Y

$$X = \sum_{i=0}^{m-1} x_i 2^i \quad Y = \sum_{j=0}^{n-1} y_j 2^j$$

$$P = X \times Y = \sum_{k=0}^{m+n-1} p_k 2^k = \left(\sum_{i=0}^{m-1} x_i 2^i \right) \cdot \left(\sum_{j=0}^{n-1} y_j 2^j \right) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (x_i y_j) 2^{i+j}$$

Arithmetic Circuits
3

So, we will look at multiplication here and we will see, how to do that in a very structured manner. So, we have what is called the multiplicand and multiplier, we have two numbers, multiplicand and multiplier. So, multiplicand is a number to be multiplied with and multiplier is a number that we have multiplied. So, in the previous video, I said subtrahend and minuend, I think I mixed up, what is subtrahend and what is minuend. So, this is the other way around.

So, minuend is this and subtrahend is what is getting subtracted from and minuend is what you are subtracting. If you do a minus b, a is subtrahend and b is minuend, so I think I mixed it in the previous video. So, anyway, let us look at multiplicand and

multiplier, multiplicand is what you are multiplying to and multiplier is what you are multiplying with.

Now, if you want to do great school multiplication, so great school multiplication is how you do it in your 2nd standard or 3rd standard, this is how you do it. If you have given two numbers, you will take the last digit of the multiplier, multiply with all the digits of the multiplicand, have it written down. Then you will take the 2nd bit, multiply with all the digits, write it down, but you will shift it by one position. Then the 3rd one will be shifted by another position, 4th one will be shifted by another position and so on.

So, we will call these, the individual bits are called partial products and this whole thing is called partial product array. So, we have an array of partial products and once you have an array of partial product, what you have to do is, you will do column wise addition. You will take 0, so that is 0 itself, 1 plus 0 is 1, 0 plus 1 plus 0 is 1, 1 plus all 0's is 1, then 1 plus 1 is 0 carry over 1, 1 plus 1 is 0 carry over 1, 1 plus 1 plus 1 is 1 carry over 1, then 1 plus 0 plus 0 is 1 and 1 itself.

So, this is the product of these two numbers here. So, I believe this is 10 plus 32; that is 42, 42 into 11, I think. So, we have this into this gives me this result. Now, in general, I can write X , so let say I want to multiply 2 numbers m and n , two numbers X and Y , let say X is a m bit number, Y is a n bit number. Then the value of X is i equal to 0 to m minus 1 $\times 2^i$, this is an unsigned number. So, it is a positional representation and this is what we have.


For Y , again it is a positional representation and that is what we have. The product P which is X times Y is $\sum_k P_k \times 2^k$, where P_k is the partial product, so that is the partial product that you are getting multiplied by 2^k . So, if you want to expand it, it eventually actually is a double summation. So, i equal to 0 to m minus 1, j equal to 0 to n minus 1, $x_i y_j \times 2^{i+j}$, all this is saying is you are taking the bit x_i , x_i has a weight of 2^i , y_j has a weight of 2^j , when you multiply that together, you get $x_i y_j$ and with a weight of 2^{i+j} .

So, the overall value that you have for the product is essentially take all the bits and multiply with each other, but with appropriate weights that is taken care of. So, let us do this on a piece of paper for decimal numbers, just to make it clear.

(Refer Slide Time: 05:37)

MODULE 52

$\begin{array}{r} 49 \\ \times 12 \\ \hline 98 \\ 49 \\ \hline 588 \end{array}$	$\begin{array}{cc} 4 \times 10^1 & 9 \times 10^0 \\ \downarrow & \swarrow \downarrow \\ 1 \times 10^1 & 2 \times 10^0 \end{array}$	$\begin{aligned} P &= 4 \times 10^1 \times 1 \times 10^1 = 400 \\ &+ 4 \times 10^1 \times 2 \times 10^0 = 80 \\ &+ 9 \times 10^0 \times 1 \times 10^1 = 90 \\ &+ 9 \times 10^0 \times 2 \times 10^0 = 18 \\ &\hline &588 \end{aligned}$
---	--	---



So, let us do this small example and I will show you, how to understand this using regular multiplication. So, let say I want to do 49 times 12. So, what I am going to do is, in the great school multiplication, you do this 2 times 49 is 98, 1 times 49 is 49 and you add it together, the result must be 588, this is what I want. The partial product interpretation says, take each and every location, multiply with each and every digit with the appropriate weights.

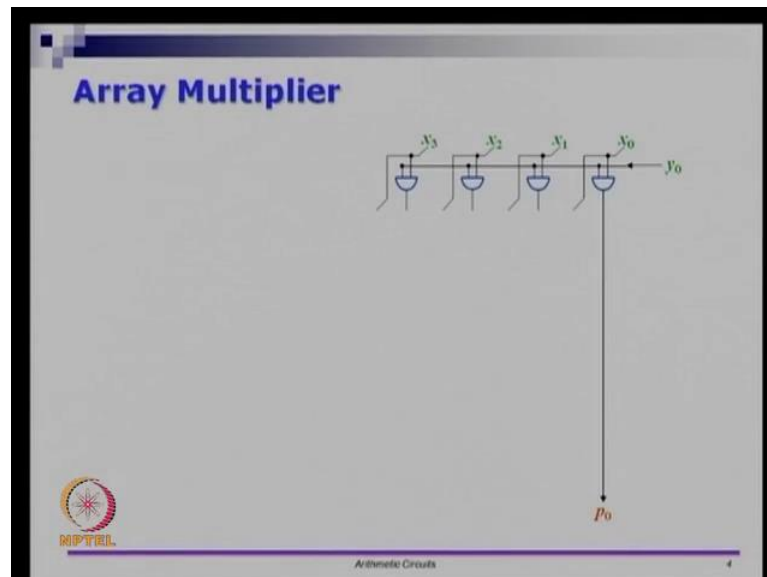
So, if you look at 49, I can think about it as 4 into 10 power 1 and 9 into 10 power 0, 12 is 1 into 10 power 1 and 2 into 10 power 0. So, the actual weights are like that and this is what I multiply, so I am going to multiply each of these pairs together. So, the overall product P is, I will take one element at a time from the top, one from the bottom and for all the 4 pairs I have to sum it up together.

So, let me take this one first, 4 into 10 power 1 times 1 into 10 power 1. So, the result is 400 plus then let say, I do these two together now, 4 into 10 power 1 times 2 into 10 power 0; that is 80. Then I am going to add, let say these two, I will multiply together; that is so 9 into 10 power 0 times 1 into 10 power 1, so that is 9 into 10 into 10; that is 90. Finally, these two together; that is 9 into 10 power 0 times 2 times 10 power 0, so that is 18. You add all of that together, that 400 plus 80 plus 90 plus 18, the result is 588.

So, it is hopefully clear that, what this product, let us get back to the slide. ((Refer Time: 07:42)) you will see that, all we have done is, we have taken each and every location \times i

and y_j , we have multiplied the bits together instead of the digits and the weights are not the 10 power something, it is 2 power something. And the weights are weight of i and weight of j together; that is all we have, it is fairly straight forward and we are going to generate a circuit which may mix this idea.

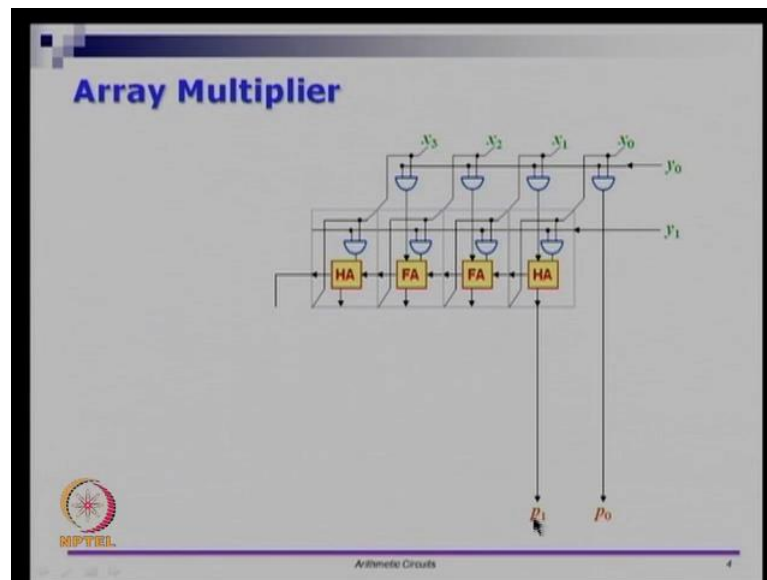
(Refer Slide Time: 08:03)



So, what we are going to do is, we are going to do something called an array multiplier. The first thing we have to realize is, getting these partial products is actually fairly easy. Here, we are not looking at digits, we are only looking at bits. So, if I take this bit 1 here, that is in the multiplier bit. If I multiply with this 1, it is fairly straight forward, what we have is, either we copy this thing directly.

So, if it is a 1, the multiplicand is there as it is, if it is a 0, it is a series of 0's and multiplying with the single bit is essentially equivalent to handing. If I want to AND two bits together that is equivalent to multiplication of the 2 bits, so a and b is the same as a multiplied by b as long as a and b are single bit values.

(Refer Slide Time: 08:52)

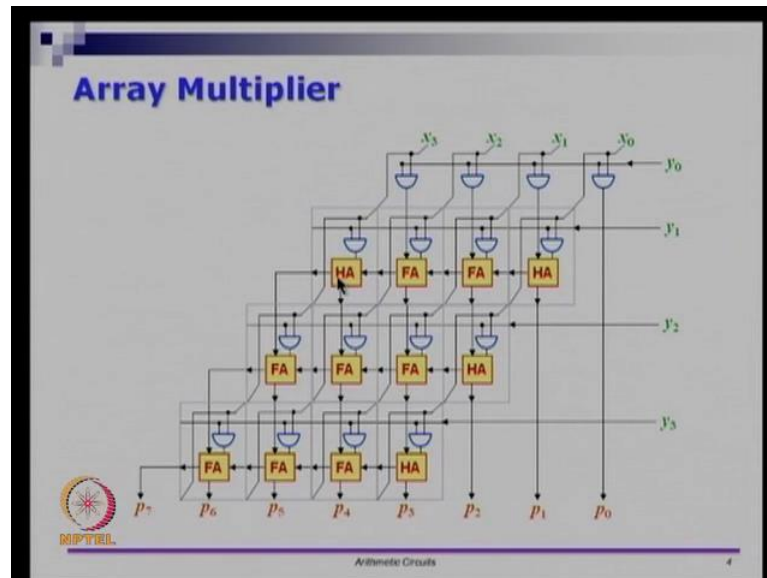


So, what I am going to do is, I am going to take let say x_3, x_2, x_1, x_0 ; that is a 4 bit number, to which I am multiplying with y_0 wherever. So, if I take y_0 , that is in the row here and in the column I give x_3, x_2, x_1 and x_0 , this is one of the partial products. So, this is the multiplier bit y_0 multiplied with the multiplicand x_3, x_2, x_1, x_0 , so the first partial products will be coming out of these and gets here.

The interesting thing is p_0 can be looked at as just x_0 into y_0 , you do not need any more calculations. Now, let us look at the next stage, you look at y_1 , you take y_1 multiply with x_0, x_1, x_2 and x_3 . You multiply all of it together, this is the same as taking the second right most digit and multiplying with the multiplicand and when you do that, that is giving you... So, that multiplication is again just a sequence of AND's, but now what we are going to do is, we going to do something slightly different.

So, now, we have two numbers and we have shifted them appropriately, we need to add these two together, so that is the addition that is done here. So, if you notice here, this is x_1, y_0 plus x_0, y_1 added together and that will be p_1 . You can and look at this circuit here, if you look at p_1 , it came from this and this product and this product, what is this product, it was this bit into this bit and this one is this bit into that bit.

(Refer Slide Time: 10:44)



So, you can see that, if I go and look at what is coming out of the half adder, the sum it is, you look at this one, it is x_1 and y_0 and this one is x_0 and y_1 ; that gives me P_1 . So, that is P_1 , you continue doing this for each and every step. So, the very first stage is just AND gates, in every partial product stage it is... So, if you put this circuit here, half adder, full adder, full adder and half adder. So, essentially what you are having is, so I get this must also be a half adder.

So, that is a mistake here, I guess this is not a full adder such it is only a half adder, so this is fine. So, what we have is, if you do this circuit here, then it is taking partial product which are coming from two levels, adding them together, generating the sum. And that sum can be added with the next partial product array; that will give you the next sum and so on, you do that for 4 steps y_0, y_1, y_2, y_3 , at the end of it what you will get is the final product.

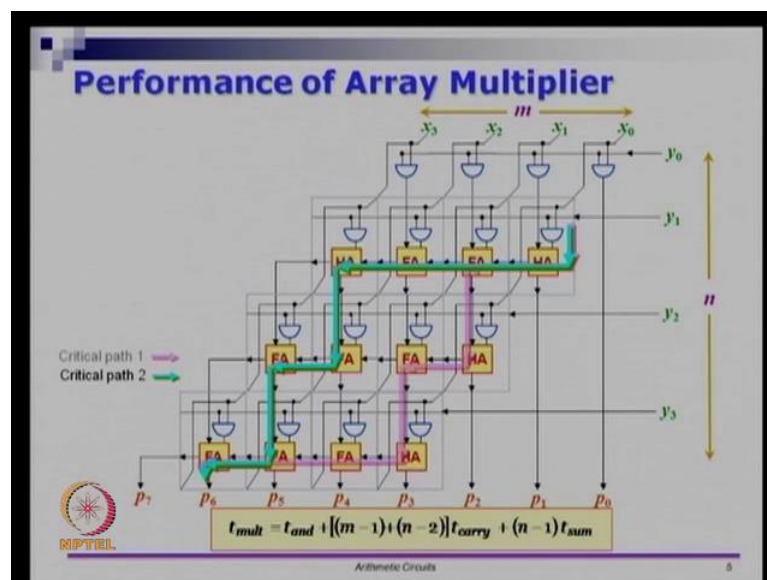
So, I hope that this is clear to you, what this one is giving is, the first partial product, this one, this AND always gives the partial products and the adder that is after that, adds of two partial product together and gives the sum. And that sum is again handed with the next partial products array that gives you the next partial sum and so on. So, the various partial products you add them one at the time and you can get the final result also.

So, this is called an array multiplier, the reason why it is called an array multiplier is, first of all it is doing multiplication. So, it is taking two numbers and multiplying them

together and it is called array multiplication, because we can do all these things. So, you can see that, it is a nice regular structure to the whole thing. If I want to another, let say I want to have x 3 to x naught multiplied with y naught to y 4, all I have to do is, take this final stage that is outer and copy it once more and this will work.

So, you can see that the whole thing is a nice bit sliced kind of structure. At every location you have a AND gate and a full adder, except for the very large stage, where it is half adder. This is a very nice and interesting circuit that you can replicate very, very easily and so, if you want to do this. So, as a circuit what we have is, we have some m bit value in this case, it is 4, we are multiplying with another n bit value also here n is 4, the result is m plus n bit value, in this case it is 7.

(Refer Slide Time: 13:27)



So, let us look at in general let say there is a m bit value multiply with that n bit value, then if you want to look at the performance of this array multiplier. So, let see how to calculate the performance, there are lots of things here it, the whole thing is combinational, you want to see what is the worst case delay that we have because of this. So, if I go and look at this path, you go and look at this path and change here, may have to go through this half adder and a full adder.

Then one half adder and full adder and if there are more and more steps, one half adder full adder and so on, it will come as those like a stair, it is like one step at a time. In the very last step, it will be this carry that is coming out of here may have to ripple all the

way through to the left most side, so that is the long path in the circuit. This is the long path, it starting from the last bit and goes all the way to the very first bit, the left most bit. So, this is one of the critical paths and there is another path, which is also quit long.

So, what happens here in this one is, you go down on this and there is a carry that is coming from this stage itself. So, this stage is the addition that you are doing for the first partial product with the second partial product; that order itself may have a delay. And once, that is settles down, it will then go down like a stair case. Essentially, it is either go down a stair case and then go all the way to the left or go all the way to the left and then go down a stair case, these two are critical paths.

And the worst case delay happens to be t of AND, which is the delay of these AND gate itself, so there is 1 AND gate here, the delay of the AND gate itself is there, along with $m - 1$ plus $n - 2$ times carry plus $n - 1$ times t sum. So, this t carry is the delay required for generating carry in the full adder and t sum is the delay required for carrying the sum bit.

So, every time you go down that is for the sum bit, every time you go to the left that is because of the carry bit. So, any you can calculate the number of steps, wherever it is green, you calculate the number of steps. So, wherever you go from left to right that will give you t carry and wherever you go from top to down; that is because of sum. You add all of it together, you will see that, there is $m - 1$ plus $n - 2$ times carry plus $n - 1$ times t sum.

So, this is you can see that, the multiplication of two numbers m and n is actually naught of the complexity m times n . The complexity is m plus n , so that is the very interesting thing. The array multiplication is taking a number of steps, which is not m times n , because you can generate the partial products in parallel and you can add numbers together, the array multiplication does not require $m n$ steps, it requires order of m plus n steps.

(Refer Slide Time: 16:32)

Signed Multiplication

Positive Multiplicand	Negative Multiplicand
$\begin{array}{r} 01110 \quad (+14) \\ \times 01011 \quad (+11) \\ \hline 00001110 \\ 0001110 \\ 0000000 \\ 001110 \\ + 00000 \\ \hline 010011010 \quad (+154) \end{array}$	$\begin{array}{r} 10010 \quad (-14) \\ \times 01011 \quad (+11) \\ \hline 111110010 \\ 11110010 \\ 0000000 \\ 110010 \\ + 00000 \\ \hline 101100110 \quad (-154) \end{array}$

Negative Multiplier: Negate multiplier and multiplicand and use the above scheme for positive multiplier.

Arithmetic Circuits 7

If you want to do signed multiplication, so let us take a simple example, let say I want to do a positive multiplicand with another positive multiplicand, there is nothing which is different. So, this is what you would have done in the array multiplier anyway, you would have generated partial products of each one and you would have added them. If the multiplicand is positive which means, so you have doing X times Y, if X is positive, then all you have to do is, in each of these location prefix is 0.

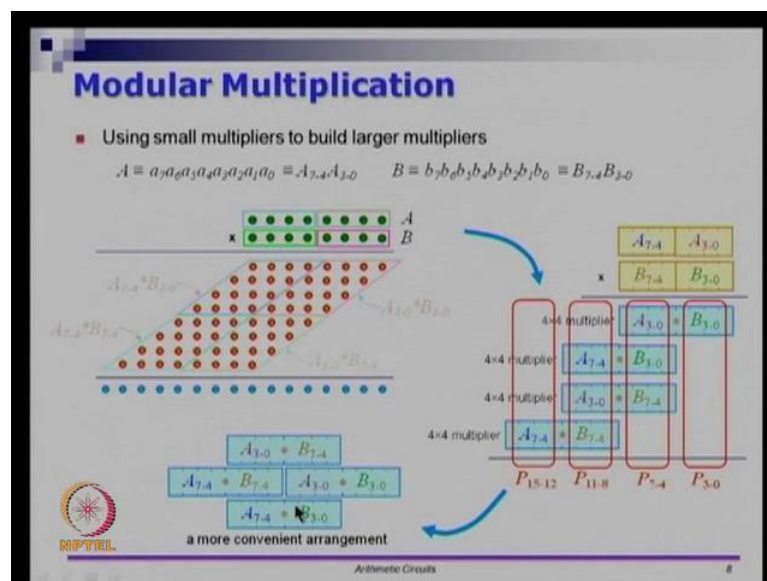
It is a positive multiplicand, so your prefix is 0 together and you add all of these together that result will be correct. So, I again you can go back and verify this, if it is a negative multiplicand however, so in this case, again we are assuming that the multiplier is positive. So, if the multiplier is positive, the multiplicand is negative, you generate the partial products as before, but you do not add them directly; instead you go and sign extend each of the partial product.

So, this partial product is 1 0 0 1 0, your sign extended it, for a sign number extension means take the significant bit and copy it till the left most one. So, here again most significant bit is 1, you copy to the left most, this is 0, so copy to the left most and so on. You add it together and interpret that as a sign number, you will realize that it is a correct interpretation. In this case this number, so this minus 14 is actually in 2's complement. So, this is not signed magnitude, it is 2's complement interpretation and the 2's complement multiplication can be done in this way.

So, this is plus 11 in 2's complement, this is minus 14 in 2's complement; the result you get is minus 154 in 2's complement. So, of course, if you have a negative multiplier, you have a problem. So, this array multiplier idea works only if the multiplier is positive, the multiplicand could be positive or negative with the signed extension. However, if the multiplier is negative, so if want to do something like a times minus b, instead treat it as minus a times b.

So, you negate the multiplier first and also the multiplicand, negate both of them and use the same scheme, it should be one of these things there. If you want a times minus b, then where b is a positive number, I want to do a times minus b, instead do it as minus a times b. Now, minus a could be positive or negative and minus a could use either this or this circuit. So, negation of the multiplier negative multiplier is not a problem, it is only a special case for the previous case that we are discusses so far.

(Refer Slide Time: 19:19)



You can also do what is called modular multiplication, instead of doing bitwise thing, you can also do something called modular multiplication, you can take a group of bits at a time and multiply them together. So, let say I have built an array multiplier, which you can take 2, 4 bit values, just that, it can take 2, 4 bit values and multiplier. If you want to build a 8 by 8 multiplier, I can do this.

Supply the lower half of a the lower half of b, get it multiplied, store the result, then you look at the lower half of b and upper half of a and store the result and so on. So, you can

do something in a very modular way. So, we can calculate completed, instead of looking at bitwise things, you can take at time or word at a time and so on and you can group the together and multiply it, and finally add them together. So, this is called modular multiplication.

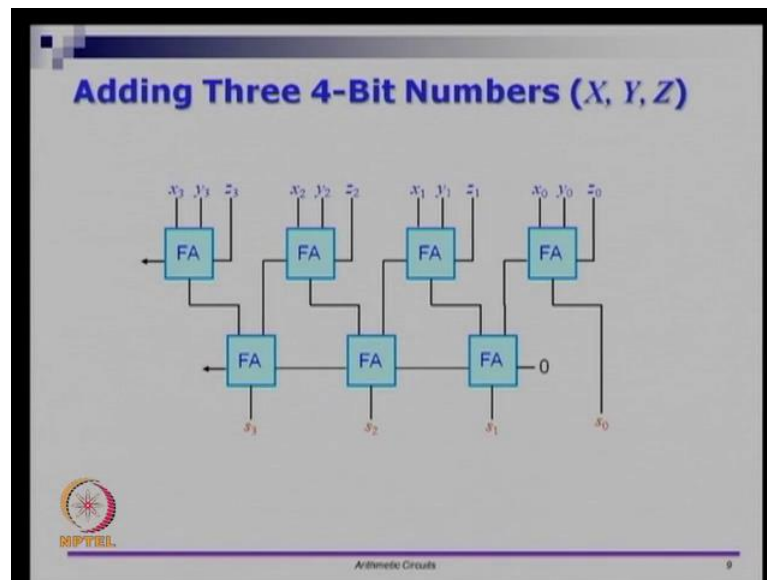
So, the modular multiplication would work like this, in the first round, you use 4 by 4 multiplier, get a 3 is 0 times b 3 0. In the second round is same 4 by 4 multiplier, but you get a 7 4, b 3 0 multiplier and once, this is over you can get these two added up. Mean while, you can use the 4 by 4 multiplier to generate a 3 0 and b 7 4 and once, that is calculated, you can use adder to add this with the top two, mean while, you can generate a 7 4 and b 7 4.

Once, that is also ready, you added with these three partial arrays, the final result will be there. So, that that is one way to do it, the other way to do that is, you generate all of them using 4 by 4 multipliers. In fact, if you use 4 parallel, 4 by 4 multiplier, instead of the same multiplier many times, you can also use separate 4 by 4 multipliers and whatever you have add it vertically with this alignment.

So, in this case, the worst case you have to add three numbers together. So, this is the number that is directly coming in, this is a partial product that will directly come in. But here we are adding three numbers and here, we are adding three numbers together, if we build a circuit that can do this, then we can do what is called modular multiplication. So, I will show you, how to add three numbers together in the next slide.

So, essentially what we have done is, we have you can conveniently do this as first do a 3 0 and b 7 4, then we do these two together and then do this to gather and add them together.

(Refer Slide Time: 21:50)



So, adding three numbers can be done like this, if I have number let us say x, y, z and if each of them is 4 bit number, if I want to add these three numbers together. Then so you take the full adder, instead of giving the carry here give the z 's, z_3, z_2, z_1 and z naught, you give those, that will give you sums and carries. So, there is a sum and there is a carry, there is a sum and there is a carry, this is a sum and this is a carry and so on.

Have the another array of full adders, where you take sum from previous stage in the carry from the next stage added together and the carry in should come from the right most side. So, this is the first level of full adders, this is the second level of full adder. So, we are not a reusing the full adder, you actually having a circuit which has 7 full adders here.

So, if you put this circuit in two steps, two full adder delays, we can add three numbers. So, instead of one complete ripple carry and what not, we have use two full adder delays and with that, you can actually get addition that. So, 1 plus 3 is 4 actually you have 4 full adder delays of the worst case. So, three numbers can be added together, instead of recurring ripple carry of n minus 1 plus another ripple carry of n minus 1, they can actually do it in 4 steps here.

(Refer Slide Time: 23:10)

Comparator – Equality Check

■ Iterative algorithm: $(x_{n-1} \dots x_2 x_1 x_0) = (y_{n-1} \dots y_2 y_1 y_0) ?$

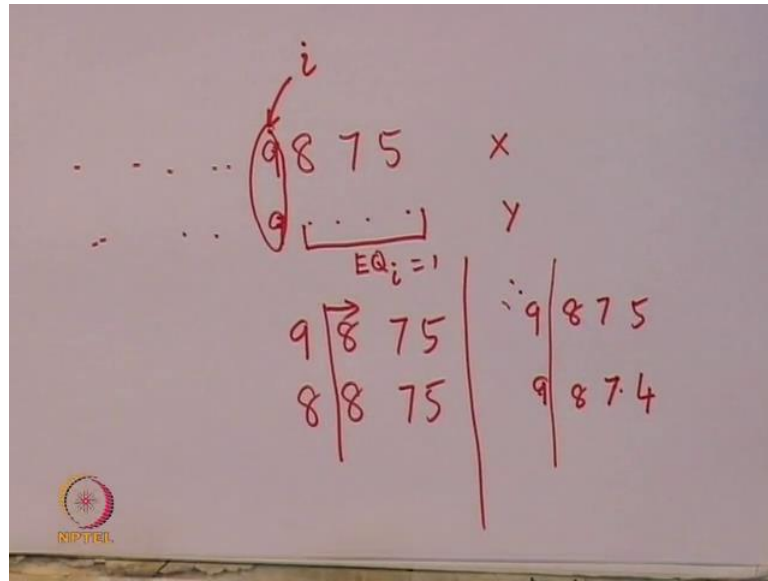
1. $i = 0; \quad EQ_0 = 1;$
2. $\text{if } (x_i = y_i) \wedge (EQ_i = 1)$
 then $EQ_{i+1} = 1;$
 else $EQ_{i+1} = 0;$
3. $i = i + 1;$
4. $\text{if } (i < n)$
 then go to step 2;
 else end;

© npptel (March 4, 2013) Arithmetic Circuits 10

So, now, let us move on to other circuits. So, comparator is a circuit, which looks at two numbers X and Y, which are of the same bit and tells you, whether they are equal or not. So, the way to do that is, you start from the right most one, let us assume that, initially the numbers are equal. If we assume that the number are equal, till we get a condition where the numbers are not equal. What we will do is, will start with i equal to 0 and we are going to run a repetitive algorithm.

We will see how to design hardware for this later, if x_i equals y_i and if EQ_i is equal to 1, which means everything to the right side seems to be the same and x_i and y_i are also matching to gather. So, you are looking at a location x_i and y_i are the same and everything to right of that were also the same. So, at this location, you still know that they are only the same. So, EQ_{i+1} is 1; however, from the right side, if they are not the same, even if the current bit is same, then they are not the same.

(Refer Slide Time: 24:23)

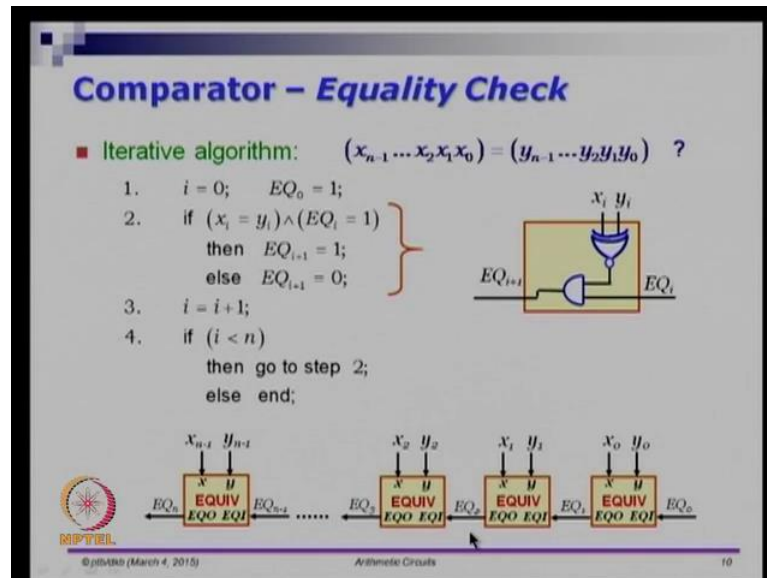


So, let us look at the example here. So, I will take the number instead of bits. Let say I am going to compare these two numbers. Let say this is the number x and this is the number y and I going to look at the location i. So, I do not know what is going to happen on in the left side, I look at positions here and here, if these two are the same, then I can say that this number is equal to this number, only if the everything to the right side, where also a match.

So, if EQ of i is 1, EQ of i is are all the digits from i to later, i minus 1 all the way down to 0, if there are equal, then it is 1 and if the current positions are also the same, then I can say that so far the numbers of the same. However, if I have a condition like this, let say 9 8 7 5 and I get 8 8 7 5 up to here things were same at this location things are different. So, which means to the left side, I should say that, the numbers are not equal or I got something like this, 9 9 in this location, it is the same, but in some other location, it was not the same. So, here it was not the same.

So, to the right side is not the same, here it is the same. Either case this number is not the same as is not, that is what a compare it as. So, that is what a seen the algorithm here, let us look at the algorithm once more.

(Refer Slide Time: 25:57)



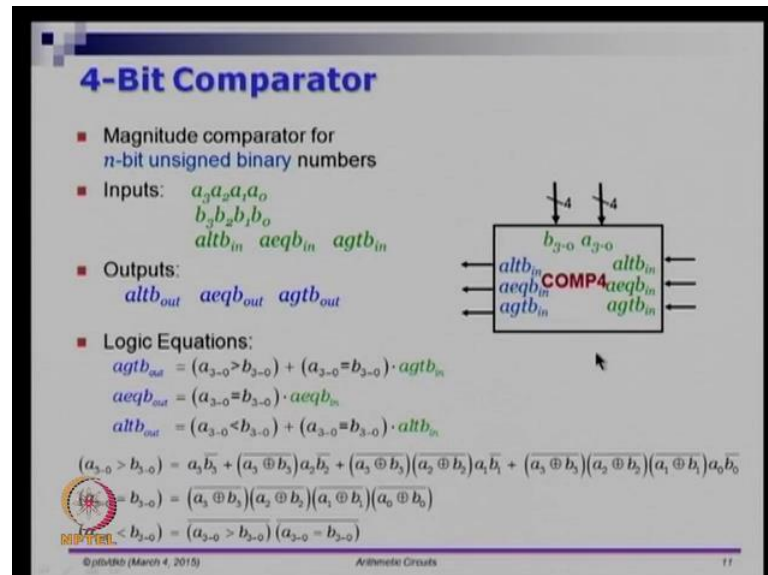
If $x_i = y_i$ and EQ_i is equal to 1, for the next stage, I can say that, these two are still equal, otherwise I will say that, they are not equal and if i is less than n go to step 1 and do this until all the i 's are exhausted. So, for one stage, I can look at whether x_i and y_i are the same by putting in an XNOR gate. So, if both are 0, this is a 1 if both are 1, this is a 1, which means at this stage these bits are equal, I have to wait for something from the right side that is EQ_i and if x_i and y_i are the same and EQ_i is also 1, then EQ_{i+1} is 1.

So, this is bits slice of a circuit that is one stage comparison if you want n numbers or n bits to be compared, I have to put them back to back. So, let us look at this example, I will put EQ_n not equal to 1, I will put 1 here, because this is a very first, this is a least significant bit. So, I do not have anything from the right side, I will put that is a 1 and x_{n-1} and y_{n-1} will be compared to each other. If they both are the same, then EQ_{n-1} will be 1, if the both are different EQ_{n-1} would be 0.

So, all we have done is, we have taken the same bits slices and repeated it again, again, which is connected it, there are no logical blocks connecting them, that is the interesting thing about a bits like circuit. The bits like circuit all there are doing is connecting the wires together without requiring extra gates. So, if I did this bits slice here, all I have to do is connect them together properly and if I go and inspect the EQ_m , it will tell me whether the numbers are equal or not.

So, this is a circuit which you can use for comparing unsigned numbers. So, there are unsigned numbers, I can compare using this.

(Refer Slide Time: 27:49)



So, a 4 bit comparator is a more general circuit, this general circuit actually can give you 3 outputs, whether a is less than b, a is equal to b or a is greater than b, this is something that you did not have in your Verilog assignment also. So, properly you did not do it in a very different way, but here on showing you, how to do it in a structural way. So, if I have 2, 4 bit numbers a and b and I want to compare them. So, and these will be the 3 bits, a less than b, a equal to b and a greater than b.

Remember, that only one of them will be on at any condition, it is not going to be different things, exactly one of these outputs which will be on. Let us assume that, these 3 we can ignore right now, let us assume that these are 3 extra inputs that we are taken in and let us ignore it just for a while. So, if I want to check whether a is greater than b, then either a[3:0] should be greater than b[3:0] or a[3:0] was equal to b[3:0] and otherwise from the input a is greater than b.

So, this I am designing of 4 bit comparator for it a slice, this 4 bit comparator may sit somewhere in the middle in a very, very large comparator. So, all I am putting is condition whereby, if I know something from the right side, how to I tell the left side whether they are equal or same or different. So, what I am going to a greater than b is true, a

greater than b out is true, if a 3 0 is actually greater than b 3 0, because this stage, if you decide this, I can look at group of digits together.

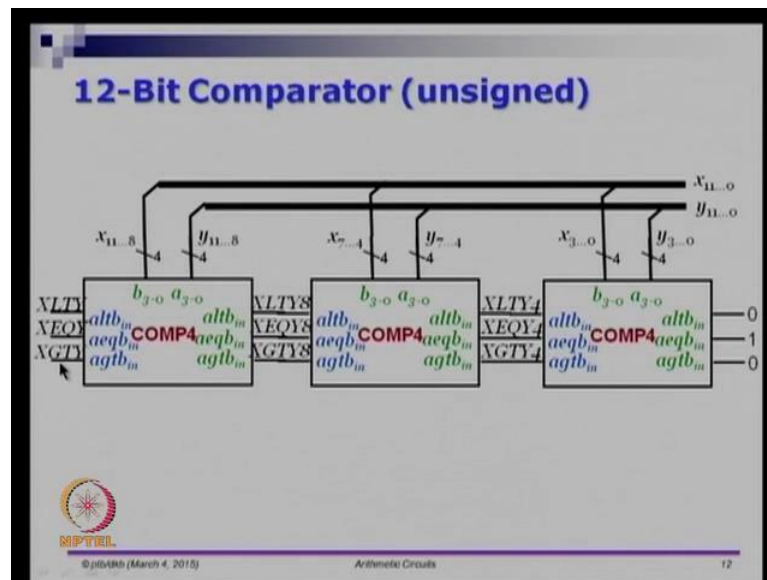
So, even in the example, I should earlier, it is not looking at 9 and 9, I would have looked at 9 8 and 9 8 together and compare it. If I decide a circuit for that, that is what this one is doing, a 3 0 is the either greater than b 3 0 or these 4 bits are the same and the current stage it looked at 4 bits, they are the same. But the right side is saying that a was greater than b, if that happens then a greater than b out is true.

So, these are single bit values, similarly a equal to b is true and only if the current stages says that, all the bits are same and the right side also says all the bits are the same so far, a less than b is true. If either the current one is, either a in the current stages less than b in the current stage for this slices or in the current slices a and b are equal, however, you got a less than b from the previous stage. So, these are the logical equations.

If you expanded you will get this Boolean equations, I suggest that you go and interpret this whole thing in English. So, there is a complicated equation here, but I suggested, we go and understand this in English to see, whether it actually means the same thing as given here. So, explain it in English, I said for the 4 bits, if I know that, these 4 bits greater than and these 4 bits or these 4 bits are equal to these 4 bits. However, from the side and gets something greater and so on.

So, for each of these, these are the 4 equations. So, for a given 4 bits a 3 0 is greater than b 3 0 can be derived using this equation, equality is using this equation and less than using this equation, I suggest that you go and understand it carefully.

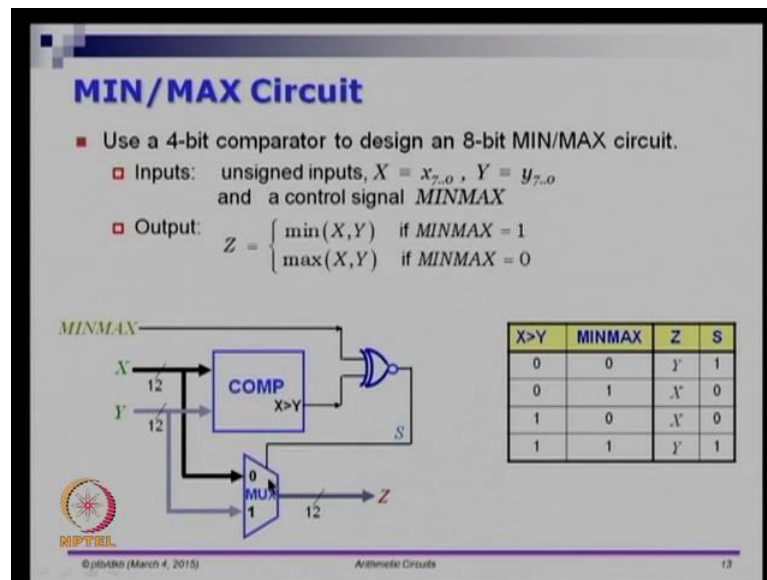
(Refer Slide Time: 31:16)



Now, if I have a 4 bit comparator, I can design a 12 bit comparator as a bit slice circuit. What I will do is, I will take 3 4 bit comparators put them one after the other, I will give 11 to 8 to both of them X and Y and the left most comparator, then 7 to 4 to the x 1 and 3 to 0 both inputs to the last one. And I take the less than output from there and connected to the less than input of this, equality output of that two equality input of and so on; that is for all these 3, for the last stage, since this is the last stage a is...

So, I cannot say that x is greater than y or x is less than y, I can start with the ground truth that x is equal to y until it is prove and otherwise. So, I cannot start with the 0 0 0, you have to careful you have to put a 1 here and will assume that, if there are 0 bits, x and y are same. Now, you get 4 bits x and y could be greater or lesser or equal in then you have 4 more bits again x and y would be greater or lesser or equal and so on at the end you have x less... So, these 3 bits x less then y, x equal to y and x greater then y, exactly one of them will be on and this is again a nice bits slice circuit.

(Refer Slide Time: 32:33)



Then I want to show you another circuit called the MINMAX circuit, a MINMAX circuit is 1, which takes two inputs X and Y and it takes a control signal called MINMAX. The output z will be equal to minimum of x comma y, if MINMAX is 1, it will be equal to maximum of x comma Y is MINMAX is 0. So, to a design a circuit like this, you would do this.

So, MINMAX is a control input, I give X and Y together and I get a single bit called x greater than y. So, this will tell me whether. So, if X is greater than y this is 1, if either X is equal to y or x is less then y let say this is a 0, then I can give this to a multiplexer, I can give both the inputs to the multiplexer. So, MINMAX should either pick x or y, even though, it is either minimum or maximum, either way it either x or y that is going to the output.

So, I will give that with this x naught condition. So, let us look at the equations here, x is greater than y is 0 which means x is actually less then y and I want the minimum, MINMAX say is 0. So, x is greater than y is 0, which means x is less then y and MINMAX is 0. So, if MINMAX is 0, I want the maximum. So, the maximum should be y. So, the select input must be 1, I have to select 1 to plus 3, if x is greater than y is 0 which means x is less then y and MINMAX is 1 MINMAX is 1 means, I want minimum.

So, minimum of x comma y, it should then the x, I want x to select x, then I must put a select line must be a 0. So, if you do this based on the comparators output and the

MINMAX control input, you will see that, this is a XOR, x is and XOR of these 2 bits, x greater than y and MINMAX bits. So, that is why it is an XOR gate. So, the XOR gate takes care of the fact that, if they both are even equal, the MINMAX will pick only one of them, either one is fine.

However, if X is less than y and you asked for the minimum, when this will be 1 and x greater than y will be a 0. So, $1 \text{ XNOR } 0$ will be a 0. So, it will pick x and so on. So, you can go and verify for all the four conditions that this in fact ones. So, this brings me to the end of module 52. In fact, this is the logical end of the course, all the course material is done with this lecture.

So, in this week, we saw several things, we started with number representation, we started with sign representation namely, one's complement, 2's complement, sign magnitude and x is b and we will look at arithmetic involving sign and unsigned numbers. So, the essential thing that you have to go with is, design a circuit for unsigned operations and with few modification, you can get it running for signed operations.

Specifically, for 2's complement is fairly easy to do, in fact that is what you will see in all the circuits, in all the processors and so on, the internal representation is all 2's complement. I also suggest that, you go and look at few PDF files and I will put in the course webpage, which has a very nice pictures to talk about, where are the different kinds of representations goes to.

Then we will look at adders namely ripple carry adder and this fast adder call the carry look ahead adder and will look at multipliers namely, the array multipliers. So, the array multiplier has time step, which is the order of m plus n and not m times n that is a very useful thing, because there is some parallelism that we are exploiting there. Otherwise, you would not get m times, we would not get m plus n , you would have got an m plus n , only if you done something in parallel.

The partial products where all generated in parallel, because of that you are able to cut down the number of steps. Finally, in the last module will look at comparators and we looked at how to bits slice them together and compare them together and so on. So, this brings me to the end of arithmetic circuits. So, what I have not done is, floating point arithmetic, I already mention that, that is slightly be on the scope of the course.

However, I will put the slides related to floating point arithmetic online, if you want to read it, you go on read it yourself. So, this brings me to the end of the course itself in fact. So, there is one more module a short video next, which actually wraps up everything that is there in the course. So, I will see you there in the next video.