

Digital System Design
Professor Neeraj Goel
Department of Computer Science Engineering
Indian Institute of Technology, Ropar
Lecture 37
Multiplication

Hello everyone, in our previous lectures we have seen how to do efficient addition and we have seen multiple ways of doing it and from those additions we have learned some new design techniques also. Today, we will extend that idea we will see how multiplication can be done in using logic gates or using Boolean logic for a digital logic.

So, in this lecture we will focus on multiplication as well as what are the issues in regular or in a Naive multiplication how we can make it faster. So, then we are discussing techniques how to make it faster than we can also see some new design techniques.

(Refer Slide Time: 01:07)



Classical multiplication

- Two step process
 - Step 1:
 - Multiplicand is multiplied with one bit of multiplier
 - Each partial product is shifted left by one place
 - Step 2: addition of partial products
- N bit multiplicand, M bit multiplier
 - Result is N + M bits
- Hardware cost (worst case)
 - M adders of size N+M
 - Can be reduced by proper sizing of adders

Example of 4 bit Multiplication

```
0010
x1101
-----
0010
0000
0010
0010
-----
0011010
```

Digital Logic Design/Combinational Circuits 88

So, a regular classical multiplication is essentially a two-step process, first multiplicand is multiplied with each bit of multiplier and each partial product is shifted by one place, so this way we got series of partial products and finally those partial products are added, so we can say there are two steps one generation of partial products and then second one is additional partial product.

While generating partial products, let us see this example of 4 plus 4 multiplication, so we each partial product is shifted by one place, so the shifting essentially means that to which place of multiplier we are multiplying it. So, now multiplier if we are multiplying with the 0th bit, so that means the partial product is at the original place, but when we are shifting, when we are multi, when multiplier is at two's place then the partial product is shifted by one place towards the left.

So, similarly when we are multiplying with this one, then the partial product is shifted by two place two places towards the left. So, this ways of all the partial products are come up are obtained and then finally we add all of them. Now, if we use a simple addition then so we also see that the total number of bits in this multiplication process it is like proven that if the multiplicand is N bit and multiplier is M bit the total bits in the sum would be N plus M bits.

So, this N plus M bits are sufficient to represent any result, for example if my multiplicand is 32-bit multiplier is 32-bit the total result would be 64-bit, will never be more than that. So, to do this multiplication using digital design or digital logic, so the way we can do is we can as we have discussed in our previous lecture so we can do addition of two partial product and then whatever is the sum then we can add that to the third partial product and then we can add it to the fourth partial product so and so forth.

So, total number of N minus 1 adders would be required and in a worst case the size of each adder would be N plus M, in average case it could be little less by properly sizing it, so for example first addition we can say that the size could be 4 plus 5, 5 bit addition can be done then 6 bit addition 7-bit addition so and so forth. But yes, the coast looks huge in worst case as well as if we try to reduce it by proper sizing then also it is substantially larger.

So, how to make it better because multiplier is like addition multiplier is also a fundamental unit used in almost all kind of designs for example, when we write C program we never think twice whenever we need to do multiplication. So, we consider that as a cheap operation, cheap means that it is probably will consume less time less area but yes looking this it looks like that my multiplication is going to be at least one order if it is N bit multiplication it is going to be at least N times more costlier than then addition.

(Refer Slide Time: 04:51)

Can we do better?

- Use carry save adders!

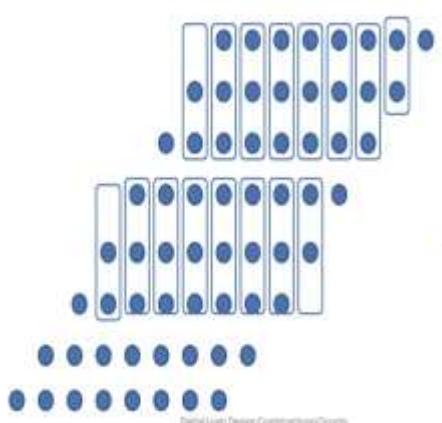
Digital Logic Design/Combinational Circuits 88

So, how can we do better? So in our last lecture we have seen carry save adders can we use that? Probably that looks like a very wonderful idea because in carry save adders has multiple operands were there and we were heading those multiple operands and we were doing this carry save thing and carry propagation was done only in the last stage.

So, but in our last lecture when we were studying about carry save adders so, this carry save adders looks like a special case and also each time we were doing some like combinations were created you know in a special way so you know ad hoc way, so can we do it in a more systematic manner? So, let us try to see this with an example of 8 plus 8 multiplications.

(Refer Slide Time: 05:43)

8x8 Multiplication example

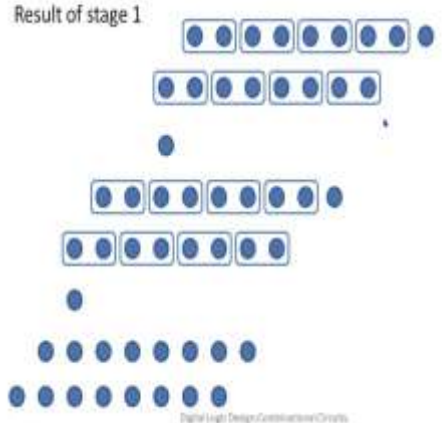


The diagram illustrates an 8x8 multiplication process. It shows two 8-bit numbers being multiplied. The top number is represented by a row of 8 blue circles, with the first 7 circles grouped into a single box. The bottom number is also represented by a row of 8 blue circles, with the first 7 circles grouped into a single box. Below these, there are two rows of 8 blue circles each, representing the partial products. The top row of partial products has 7 boxes, each containing a pair of blue circles, representing the products of the first 7 bits of the top number with the bottom number. The bottom row of partial products has 8 blue circles, representing the product of the 8th bit of the top number with the bottom number. The APTEL logo is in the top right corner. A small video inset in the bottom right shows a man speaking.

Digital Logic Design, Combinational Circuits

8x8 Multiplication example

Result of stage 1

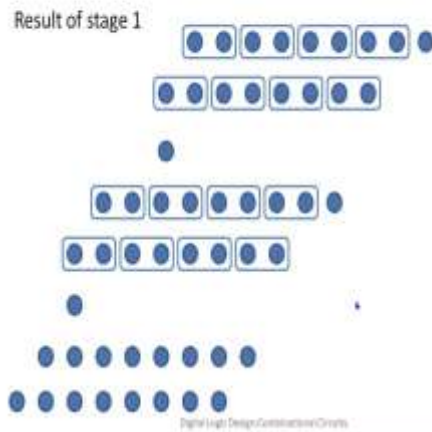


The diagram shows the result of the first stage of an 8x8 multiplication. It features two rows of 8 blue circles each, representing the partial products. The top row has 7 boxes, each containing a pair of blue circles, representing the products of the first 7 bits of the top number with the bottom number. The bottom row has 8 blue circles, representing the product of the 8th bit of the top number with the bottom number. The APTEL logo is in the top right corner. A small video inset in the bottom right shows a man speaking.

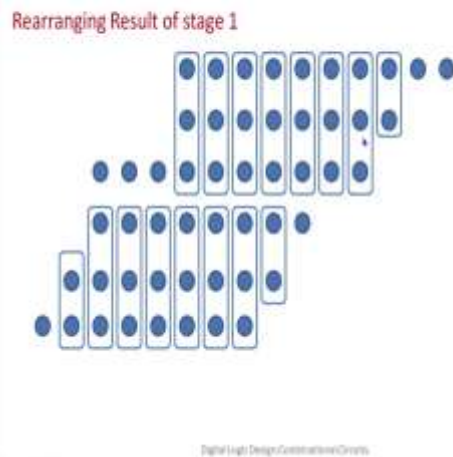
Digital Logic Design, Combinational Circuits



8x8 Multiplication example



8x8 Multiplication example – stage 2



So, in an 8 plus 8 multiplication this carry save adder will work only in the second step, the first step where we are generating partial products carry save adder is not going to help there, but once partial products have been generated after that it can help in addition fast addition of all of them. Now, let us see we are doing it goes 8 plus 8 multiplication and each of these dots represent 1 bit of the partial product.

So, this first row will represent first partial product, so because it is 8 plus 8-bit each bit would be so each bit of my multiplier would be multiplied with the multiplicand, so these 8 bits I will reserve this 8 bits, similarly when the second bit of multiplier would you multiplied with multiplicand then this row would be obtained.

So, similarly I would have such 8 rows and each of these 8 rows represent my 8 partial products. Now, I need to add all of them, to add all of them and also I would like to use carry save adders, so to use carry save adder what should I do? I have to identify that which bits I can add in parallel. So, we see that here these 2 bits are actually of same order or same power or denotes same place.

So, we can add both of them and similarly these 3 can be added because they again belong to the same place, so similarly all these 3 can be added together. So, like this you can also add all these 3 bits in this particular these 3 rows and these we are leaving because the fundamental method is the fundamental method which we are using to create these groups is that if number of rows or number of dots number of wires I can say are more than let us say multiple of 3, then I can make group of 3.

So, what if it is more than 3 more than let us say for example, here it is 8, so we are leaving these 2, so that we can use them in later steps. So, this way we are creating all the groups and the groups are created based on equal powers. Now, after doing this grouping now each of them this would require an half adder this would require an half adder and these bits would require some full adders.

Similarly, these bits would require full adders, so all of them would be added and every stage these 3 would be now for example for this addition I will generate carry and a sum, sum would be for this place and carry would be for this place. So, this way what we can see that if we see the result of this first stage, let us call it at the stage, so in the first stage, we have grouped these many groups and the result of this group would be this stage.

So, for example there were two dots here, now this is the sum these they carry part, carry part would be in the in the next group. So, similarly a 4 all others now there were 3 bits here, so these 3 bits converted into 2 bits, so this say this is the result of stage 1, where we are able to eliminate these dots, so essentially what a carry save adder is doing for us that it is removing, so instead of adding 3 numbers now it will reduce them to 2 numbers sum and carry will carry would be in the next order and sum would be in the same order.

So, let us try to see it closely enough we will keep flipping the slide from previous slide to this slide to understand how this step has been done. So, you see that there were 3 bits here, so these 3 bits the sum would be here and the carry would be in the in the next group. So, some is here and carry is here, so similarly for this group sum would be here and carry would be shifted to the next place.

So, sum would be here carry would be shifted to this place, so to make things easier we have shown in 2 different rows, but essentially they were old belonging to the same row. The only thing we have to understand is that each sum each of the sum each of these full adder will return sum and carry, sum would be would result in the same column, but carry would be given to the next column.

So, similarly for these also the sum would be here and carry is here sum is here carry is here, so 1, 2, 3, 4, 1, 2, 3, 4, so total 8, 1, 2, 3, 4, so total 8 additions were there because of those additions we are essentially able to eliminate one whole row, so we are able to eliminate this row, we are able to eliminate this row, so after one stage of addition instead of 8 rows now we have 6 rows. So, now we can reorganize this result, so for example, we can shift this dot from here to here we can shift dot this dot from here to here, so we will reorganize it looks something like this.


So, we will have total of 6 rows and now again we can go ahead and group them, so we can find out in this stage 2, after rearranging how we can do additions, so we can group these two, we can group these three, so you will see that we are able to do sum 8 additions here the same 8 additions we can do here also. So, we are leaving wherever single loads are there because they can be added in the next stages. So, again after doing this addition total number of rows would reduce from here, the number of rows are 6, so these 6 rows would reduce to 4 rows essentially.

(Refer Slide Time: 12:27)

8x8 Multiplication example – stage 3

Result of stage 2


Digital Logic Design/Combinational Circuits



8x8 Multiplication example – stage 3

Rearranging result of stage 2

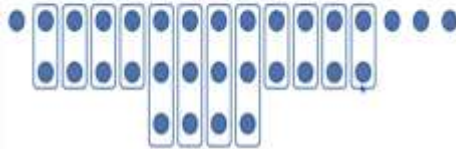
Digital Logic Design/Combinational Circuits





8x8 Multiplication example – stage 4

Rearranging result of stage 3



8x8 Multiplication example – stage 5

Result of stage 4



8x8 Multiplication example – stage 5

Result of stage 4

Adding these using carry propagation adder

Overall reduction in area and delay!

This is called Wallace tree Multiplier.

Digital Logic Design, Computation and Circuits 11

So, we have removed all of them similarly this particular row is eliminated now this dot we can move upside, so if we rearrange them after rearranging, so after rearranging this will become 4 rows. Now, this is the result of stage 2, after this stage 2 again we can reapply our carry save additions, in this carry save addition we can again form this the of group of 2 this is the group of 2 this is the group of 2 and now these are the groups of 3.

So, these additions we can go ahead and perform and after this addition now these 4 rows would be reduced to 3 rows. And these 3 rows again we can rearrange so that we can perform the addition again. So, what you are observing that there are multiple stages we are doing multiple additions and after each addition some rows are getting eliminated.

So, after this stage 4 again we can go ahead and perform these additions, we can form wherever we can form groups. So, after forming additions like now these 3 rows has been converted into 2 rows, this is the result of stage 4. So, after stage 4 now this is essentially the last stage and this last stage what we had to do we will not use carry save adder but propagation has to be done.

So, each time, so for example addition of these 2 bits would be given would be giving carry to this this addition and similarly addition of this carry, additional these 3 bits would be given as input to this addition, so essentially carry would propagate now, so in this last stage, it is a carry propagate adder. So, the design of adder would be like a propagating carry where every stage every set of bits would be giving input in the next set of bits.

So, now the delay of this 5th stage is essentially equal to a carry propagate adder, carry propagate adder. Now, if we see the overall delay, so because before these they were 4 stages, these 4 stages were consuming 1 adder delay, the worst case was 1 adder delay, the total number total amount of delay I can say here would be essentially equal to approximately order of N . So, because of the previous 4 stages now I need not to do any kind of addition for these 4 bits, for rest of the bits, yes I need to do addition.

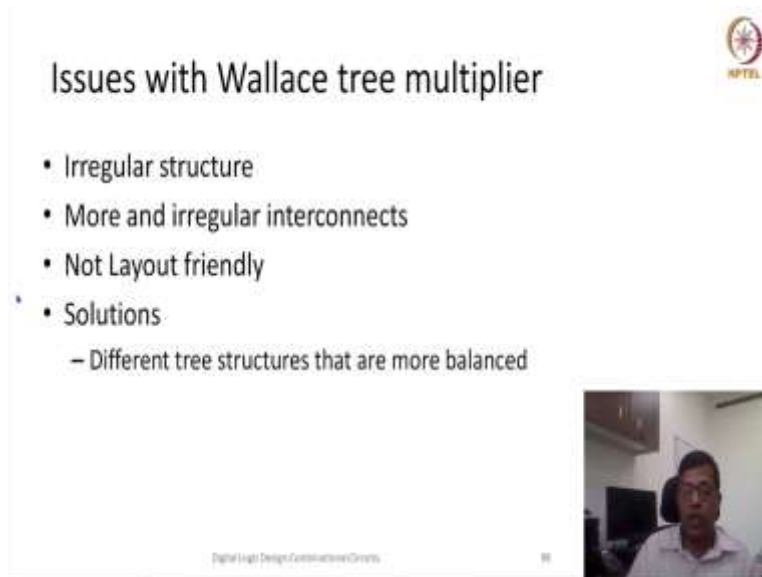
So, if 8 cross 8 multiplication is resulting in 16 bits, so I would require this addition 12 full adder additions here and the 4 additions has been done in 4 different stages, so the total delay is of the order of N , or basically N plus M . So, if I have a multiplier which is, which has multiplicand of size N and multiplicand of size M , so the total delay is going to be N plus M of the order of N plus M .

So, what would be the impact of number of adders? Number of adders would also reduce substantially because we are playing smart here wherever addition wherever number of bits are sufficient then we are adding otherwise we are not adding, we are also leaving to the next stages so that we will have a sufficient number of bits which can be combined together. So, this forms a very efficient technique of multiplication this is called Wallace tree multiplier.

So, in this Wallace tree multiplier, the group of additions are created in such a way that the number of stages are less, what are the number of stages? The number of stages are approximately $\log n$, so how do you come to this fact of Logan because after every stage ideally we should be able to remove or basically reduce the number of rows into 2 by 3.

After every stage because the number of stages are reduced to 2 by 3, so if my addition is very very large, so the number of bits are very very large. So, the number of stages would be low of the order of $\log N$. And then finally in the last stage, we are going to add them in a in a carry propagate manner, so the delay is always going to be of the order of N full adders. So, this looks good.

(Refer Slide Time: 17:44)



The slide is titled "Issues with Wallace tree multiplier" and features the APTEL logo in the top right corner. The main content is a bulleted list:

- Irregular structure
- More and irregular interconnects
- Not Layout friendly
- Solutions
 - Different tree structures that are more balanced

In the bottom right corner, there is a small video inset showing a man speaking. At the bottom left of the slide, the text "Digital Logic Design/Combinational Circuits" is visible, and a small number "98" is at the bottom right.

But there are still some challenges or some issues in this Wallace tree multiplier or multipliers like them. So, first the issue is that you have a very irregular structure. So, if we have seen that every time you require to do addition, so you do not know so basically you have to lay out you have to fix all these bits and you should know in every addition that which particular bit you would like to connect to.

So, you would require N number of adders, different number of adders in every stage and every stage would require different kind of interconnect also. So, it is not very regular, it is not that 1 bit is always going to the next adder, so the overall technique is not set to be layout friendly, layout friendly means that it is not rectangular in nature, so in some particular stages number of adders are more in some other adders are less.

So, to this irregularity problem and layout issues and irregular interconnect issues, some people have provided our different kind of a tree structures, like in Wallace tree the structure is different, there are other trees where the structure is more uniform, so that the number of additions could also be minimized number of full adders requirement can also be minimize people have, so this Wallace has been proposed some 50 years back.

So, in last in last 50 years people have done meticulous calculation that which particular creation of which particular group will lead to minimum number of full adders or minimum number of

half adders. So, with those calculation number of full adders could be minimize plus integrate can also be minimized by creating more a regular structure rather than a haphazard structure where any bit could be added to any other bits.

So, with those yes, this Wallace are still the kind of fastest multiplier or a tree base multiplier I would say is tree base multiplier will carry save adder base multipliers are still the among the fastest multiplier we have. So, in any kind of a VLSI design the or a digital design these tree multipliers are going to be the fastest one.

So, do we have some other alternatives? Can we make it even more structural structured or even more systematic so that it can be done in a more peaceful manner?


(Refer Slide Time: 20:33)

Array multiplier

- Combining generation of partial product and summation
- Identical cells in each row
- Example : 5x5 multiplication
 - Inputs $X (x_5 x_4 x_3 x_2 x_1 x_0)$ and $A (a_4 a_3 a_2 a_1 a_0)$

Digital Logic Design/Combinational Circuits

APTEL



Array multiplier



- Combining generation of partial product and summation
- Identical cells in each row
- Example : 5x5 multiplication
 - Inputs X ($x_4 x_3 x_2 x_1 x_0$) and A ($a_4 a_3 a_2 a_1 a_0$)



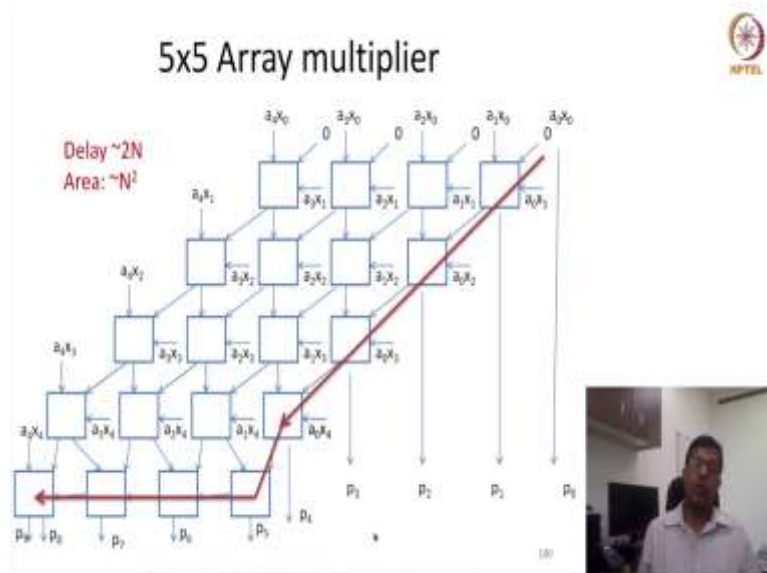
Digital Logic Design/Combinational Circuits

39

So, for that let us try to understand one more multiplier which is and which is called an array multiplier or is also called combinational multiplier. So, here the beauty of this method is it is not saying that give me the partial products then I will add, so it will it has combined the generation of partial product as well as summation. So, each row is essentially doing the partial product computation as well as doing the multiplication.

So, let us try to understand this particular multiplied with a 5 cross 5 multiplication example, so and the example is generic enough that you can easily extend it to any kind of multiplication. So, we are considering 5 bit input X and 5 bit input A, so this X is x_0, x_1, x_2, x_3, x_4 , similarly A is a_0, a_1, a_2, a_3, a_4 . Now, this X is being multiplied with A, so, my multiplier is X and A is my multiplication.

(Refer Slide Time: 21:35)



Now, since X is my multiplier A is my multiplicand, so in the first stage here also I am trying to use the same word stage, but the meaning is slightly different here. So, in the first stage, I am adding 2 partial products, so $a_0 \times x_0$ I am doing an AND gate and $a_0 \times x_1, a_0 \times x_2, a_0 \times x_3, a_0 \times x_4$, so this is the first partial product, the second partial product is $x_1 a_0, x_1 a_1, x_1 a_2, x_1 a_3, x_1 a_4$, it would come later let us see.

So, we are using these 4 full adders, so these are full adder, this is one operand this is the second operand and carry is 0 here, so carry is 0 and we all know that in whenever we add this first stage $x_0 a_0$ will always form the first bit of the product, so there is no doubt about that, so we can simply keep $x_0 a_0$ as p_0 . And since the addition of these two is also there $x_1 a_0$ and $x_0 a_1$ addition of these two will also result in p_1 , so this is also pretty clear.

So, what 4 other additions 4 other results p_2, p_3, p_4 et cetera we need to add more. so, here using one layer of addition we are able to add two partial product, partial product that results because of x_0 the partial product that result because of x_1 , these two has been added using one row or one layer of my full adder. Now, similarly the result of this full adder layer there would be two results one would be sum or other would be carry.

Now, I can add this sum and carry to another layer where the third input is coming from the third partial product, third partial product means multiplication of x_2 with my multiplicand, $x_2 a_0, x_2$

$a_1, x_2 a_2, x_2^2 a_3$, so you remember in our previous layer $x_1 a_4$ was missing, so this $x_1 a_4$ was just left like this and this $x_1 a_4$ is added to the adder in the in this new layer, the second layer. So, in the second layer also we will have we are adding the result of previous partial addition, we have done the partial addition and the result of partial addition is now added to the new partial product.

So, this partial addition you see the carry is also generated after every addition, so that is why we are using full adders here. So, whatever is the carry from this adder is added as a carry here, whatever is the result of carry here that is added to the to the carry input of that is connected with the carry input of this particular adder full adder. Similarly, a result of this carry result of this sum, carry of this addition is given as a input C into this addition.

So, all of these full addition will form the second layer. Now, similarly a third layer would be there, you also notice that as a result of second layer we also have got or we have also generated second bit P2 bit over result. So, similarly the third layer of the fourth partial products so multiplication of x_3 with my multiplicand would be added in this particular layer.

So, here also like our previous layer like whatever now these vertical arrows represent these vertical arrow represent the overall addition done so far and the slanted or distilled arrow represent the carry through the previous full adder, so all of them would be added in this stage and so this is the by now we have added 4 partial product, product with x_0 , product with x_1 , x_2 and x_3 .

In the next stage we would be adding the fourth or the last partial product to the final sum. So, in this last partial product you see the $x_0 a_4$ is the input here and whatever is the carry from the previous adder is given as a carry input all the vertical lines are the sum from the previous layers. So, you still see that $x_3 a_4$ the component which was left from the previous partial product would be added as a vertical arrow or a as a one of the input.

Now, what about the some of these final ones? Now, it is a last interesting case the last stage and also the most interesting one, here this is like in Wallace tree multiplication this is also a carry propagating stage, here you are you whatever result we are receiving here, so for example the result of this because of space constraint we have to shift make these arrows little more not so

straight as they were here, so whatever is the carry of this stage is coming as a carry input of this the sum is essentially a vertical arrow, so the sum from this stage is given as input here.

Now, remember see the carry which is out, carry out from this particular full adder is given as an input to the next one. So, this is the carry in from this full adder this is sum from this adder and the resulting carry is given as an input to the next full adder. Similarly, here also resulting carry is given as input to next full adder, $x_4 a_4$ is also given as an input to this final full adder

And you will also see that $P_4 P_5 P_6 P_7 P_8 P_9$, so $P_5 P_6 P_7 P_8 P_9$, so these are also the final outcome of the last layer. So, in this way this looks like a very regular structure where we clearly know how many full adders would be required, what is going to be the delay of this particular adder, and how they would be connecting. So, while designing you can see because the structure is so nice that I can create these partial additions and I can create N minus 1 stages of these partial additions.

And finally this particular stage could be different where we are propagating carry where delay could be slightly different, but these also quite a regular structure. So, this particular array multiplier or a combination multiplier is more effective solution whenever a regularity is required. Here my overall hardware structure will also be sort of a rectangular or even a square structure which is very low friendly.

What about area and delay? So, here I can see clearly the delay is order of $2N$, you can see it is not exactly $2N$, but we can calculate from here the adder delay of this the if we see the longest path or the most critical path here you will see it is coming from here, this addition this addition, so this is the longest path, this addition then this addition the output is going to this addition and then output is going to this addition then output is going to this adder, this output is going to this adder this output is going to this adder so on so forth.

So, the total number of full adders in my critical path is $2N$ minus 1, total here it is 1, 2, 3, 4, 1, 2, 3, 4, the total number is actually 8 here. So, that means whenever whatever is the size it is always going to be $2N$ minus 2, why? Because the first bit we will always get from here $x_0 a_0$ and the last bit final bit we will get as a carry of my final adder. So, this way whatever is let us say this the number of operands are N and M that the delay is going to be N plus M minus 2.


And similarly if you see the area the area is also predictable, so here you can say in my initial stages $4 \times 4 \times 4$, so basically N minus 1 square that would be the number of adders which would be required in adding partial products and in final carry propagation stage again N minus 1 numbers of address would be require, so the total number of adders are all the order of N square.

So, this particular array multiplier also have some additional interesting properties that I can easily divide this this particular addition in two different stages, one, so here you see the delay of this particular variable where we are saving the carry we are not where we are adding the partial products we are not calculating the final some, the delay of this these 16 additions are equally good for equally same equal to the delay of my last stage.

So, I can divide this adder into two different stages, one where I am adding the partial products and the second one, where I am propagating the carry and calculating the final sum. So, this way this creating this multiplication as a two-stage multiplier also help in something called pipelining which this pipelining will understand little later.

But it would help us in breaking this multiplier into two different stages which are both almost equal in delay. So, with this we can almost closed over multiplication method. So, there are many multiplication method, but I have chosen these two so that we can you can understand some of these techniques which can be used for multiplication.

(Refer Slide Time: 33:12)



Summary

- Efficient design of multipliers
- Are these multipliers valid for unsigned as well as signed numbers?
 - No, for signed number, partial products need to be signed extended
 - Need some modification in the design

Digital Logic Design/Combinational Circuits 101

Now, as a summary we have seen in this lecture how to design efficient multipliers, we have seen two different methods, one is tree based method where tree of carry save adder was created and finally a carry propagation layer would be required to add all of them and the delay for that particular so the number of stages there were around low and the delay was also the order of N , while we also seen this very regular structure like array multiplier or a combination of multipliers in this combination particular array multiplier the number of the delay is slightly more, but it is more a regular structure and also be divided equally into two parts.

So, with this we can almost close this lecture, but let us ask one more question to ourselves that when we have studied multiplication binary multiplication, so at that time we have seen that these if we multiply $2N$ bit numbers and these both of these $2N$ bit numbers are represented using two's complement then and a result is also N bit result then we can say that the result is always going to be correct.

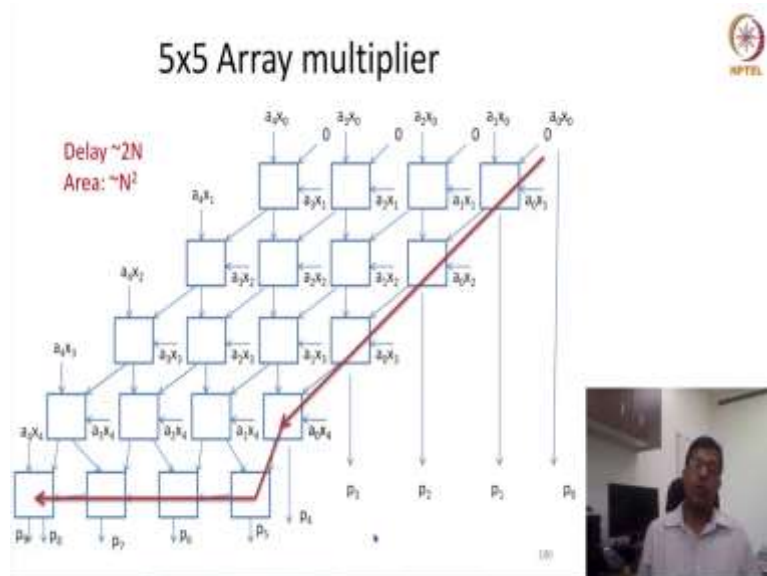
But in this particular exercise in today's lecture we have seen the result is of size $2N$, so the question is whether this result of $2N$ will always give correct result in case of unsigned numbers as well as signed numbers? Answer is no, in case of unsigned number it will always give correct results but if we follow these techniques as is as is means that the way it has been described in this lecture if we simply add those partial product in the same way, then sign number will not give correct results, there would be certain modification which is required.

So, I am not going to discuss those modification in detail, but let us give a very quick intuition, so essentially let us say my multiplicand was a negative and multiplier was positive or negative it does not matter, so if multiplicand was negative each partial product the first partial product, so let us consider 4 plus 4 multiplication, in case of 4 plus 4 multiplication the first number after multiplying with the 0th bit of multiplier is a 4 bit number.

So, now this 4 bit number tell me whether this number is a positive number or a negative number. Since, the result is of 8-bit and partial product is a 4-bit what are the rest of the what are we going to assign to the rest of the bits? So, because my multiplicand was negative I have to sign extend each of these partial product. If I do sign extension of each of the partial product, then all of the techniques which we have discussed today will work in a same way.

The only thing we had to do additionally is we have to have this sign bit or sign extension, we have to have additional full adders which will take care of the sign bit which will take care of this which will do sign extension and we should also the signs would also be great along the path. There would not be any impact on the delay.

(Refer Slide Time: 37:02)



So for example, if we see this last one, so let us say there is a another addition there is another addition here, so still the worst case path is going to be $2M$. So, there is no impact at least on array multiplier there is going to be no impact if we have the additional adders in this lower half, in this upper half, triangle also they would be half adders full adders and these full adders will create some additional results and but the delay is going to be still same. So, with this we can close today's lecture and I will say thank you very much.