

**Architectural Design of Digital Integrated Circuits**  
**Prof. Indranil Hatai**  
**School of VLSI Technology**  
**Indian Institute of Engineering Science and Technology, Shibpur, Howrah**

**Lecture - 35**  
**Multiplier Architecture ( Contd. )**



Hello everyone. So, welcome back, again to the course on Architectural Design of IC's. So, in the last class we have seen that array multiplication architecture. So; why we use array multiplication? Because it has or it gives you the; that means, regular structure for implementing any multiplication operation in compared to the three based multiplier architecture, but it has the disadvantage of that means, it has to has the carry is rippled. So; that means, the delay or; that means, this particular array based multiplier architecture they are slow in operation ok.

So, and apart from that we have seen that that Baugh-Wooley multiplier can be; that means, for unsigned multiplication we have seen that array multiplication architecture. So, that sign multiplication architecture or which is nothing, but the Baugh-Wooley multiplier. So, that means that sign multiplication architecture that also can be represented in terms of array multiplier architecture. So, that we have to; that means, we will see in today's class ok.

(Refer Slide Time: 01:31)

**Baugh-Wooley Two's Complement Multiplier**

$$\begin{array}{r}
 \begin{array}{cccccc}
 & & -a_4 & a_3 & a_2 & a_1 & a_0 \\
 \times & -x_4 & x_3 & x_2 & x_1 & x_0 & \\
 \hline
 & & a_4x_0 & a_3x_0 & a_2x_0 & a_1x_0 & a_0x_0 \\
 & + & a_4x_1 & a_3x_1 & a_2x_1 & a_1x_1 & a_0x_1 \\
 & & a_4x_2 & a_3x_2 & a_2x_2 & a_1x_2 & a_0x_2 \\
 & & a_4x_3 & a_3x_3 & a_2x_3 & a_1x_3 & a_0x_3 \\
 & + & a_4x_4 & a_3x_4 & a_2x_4 & a_1x_4 & a_0x_4 \\
 & & a_4 & & & & \\
 & & x_4 & & & & \\
 \hline
 \end{array} \\
 \begin{array}{cccccccc}
 & 1 & & & & & & & & \\
 & -2^4 & 2^3 & 2^2 & 2^1 & 2^0 & & & & \\
 \hline
 \end{array}
 \end{array}$$

So, if we see that this is the Baugh-Wooley two's complement multiplier; that means, all the multiples which I have to consider in Baugh-Wooley two's complement multiplier system ok.

So, as this particular operations are different in corresponds to that unsigned multiplier architecture. So, then if I have to; that means, if I in unsigned multiplication what was happening? So, this basically this operation they are very much common so; that means, whenever I am adding at that at that time; the corresponding this 2 x 0 or a 1 x 1 or a 0 x 2 they that I have already; that means, added ok.

So, but here you see the some of this multiples they are a 4 x 0 bar ok. So, here like a 1 bar x 4 and a 4 and x 4 here I have to add; that means, this these are the extra terms which are coming at this particular position. So this addition of this extra overhead because of this; that means, two's complement multiplier that we have to consider in this particular case.

(Refer Slide Time: 02:57)

### Modifications in 5 x 5 Array Multiplier

- Shaded cells indicate additional cells for Baugh-Wooley
- When sum logic longer delay than carry logic, critical path goes through diagonal then along last row
  - Can reduce this by causing some sum signals to skip rows
  - These are shown as curved arcs in figure

IIT KHARAGPUR
 NPTEL ONLINE CERTIFICATION COURSES

So, how we can do? So, in this array multiplier, for this Baugh-Wooley multiplier, so, for that reason; so this was the initial array multiplier for this unsigned multiplication ok. But for what modification I need to make it this; that means, make it useful for Baugh-Wooley multiplier; that means, I have to because of this extra overhead or extra terms which are coming at this particular position, so; that means, I need extra cells to be filled for implementing this array based Baugh-Wooley multiplier.

So, what is that? That means, at this particular for this just P 0, P 1, P 2, P 3, P 4 position I am getting 2 extra terms which are a 4 and x 4 ok. So; that means, in initial; that means, multiplier I am having only 4 stage here, but here I need another extra of this element processing element which will consider this a 4 and x 4 too which will be added with the previous terms and then that will produce the P 4.

Apart from that what I need? I need at the 2 to the power 8 position or this P 8 position what I need? I need to add then again 2 extra terms which are a 4 bar and x 4 bar ok. So; that means, at this particular position here you see at this particular position I need to add this a 4 x 4; a 4 bar and x 4 bar which are coming through this which is basically I need one extra processing element block which are of coming; that means, which are taking the input of a 4 bar and x 4 bar and then that is basically producing the P 8 terms.

Then again there is another extra overhead terms for this P 9 that is 1 ok. So, then that 1

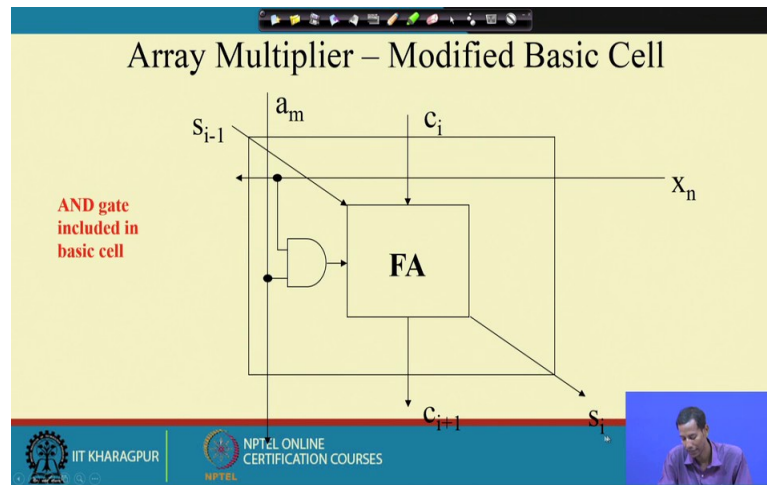
again that will be added with the carry which is being generated from this P 8 block. So, this carry is being basically the carry for this a 4 x 4 a 4 bar and x 4 bar one carry and the carry which are being forwarded from the previous stage the which is that P 6 sorry for P 7 stage ok. So, that carry also has to come to this P 9 stage ok. So, they can then again that will be also passed that p; P 7, P 8 and then it will come to the P 9 stage those carry also has to add and then I will get the final P 9 over here ok.

And then what other changes I have to do the terms which are typically coming over here they are different. So, initially it was not like a 4 x 0 bar it was a 4 x 0, but here I have to put this as a 4 x 0. So, the same as a 4 x 1 bar; so, instead it was a 4 x 1. So, those type of modification also I need to do to; that means, to make it 5 cross 5 array multiplier architecture for two's complement Baugh-Wooley multiplier.

So, here you see that shaded cell indicate additional cells for Baugh-Wooley multiplier when sum logic is longer delay, than carry logic critical path goes through diagonal then along with the last row. And can in this can reduce this by causing some signal to skip rows and these are shown as curved arcs in figure. So; that means, what I can do or; that means, what is basically happening over here; that means, here what I am doing? I have increased the corresponding critical path over here; that means, at that time it was 4 and then 4 here, but here now what it becomes; 5 and then here also 5. So that means, now the corresponding critical path becomes 10 full adder cell. So, initially it was what I said if I use; that means, 5 cross 5 array; so, the total critical path delay will be m minus 1 plus m minus 1; that means, 4 plus 4 that was 8 full adder cell.

So, if you see that 4 over here and 4 over here. So, 8 full adder cell maximum, but here now it becomes 5 and 5; 10 full adder cell because of the extra modification which I need for this modified array multiplier architecture ok.

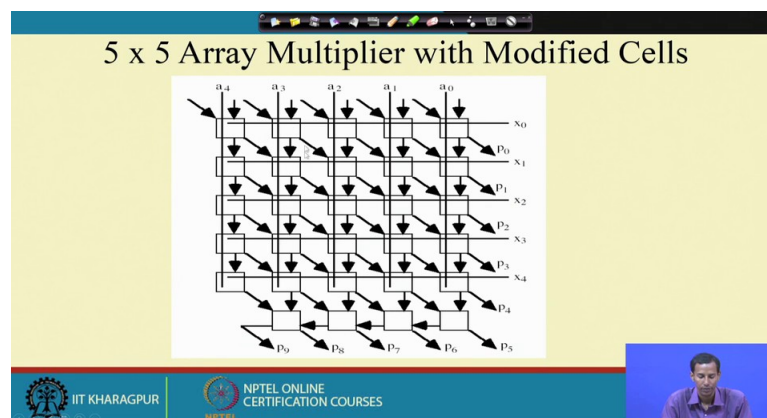
(Refer Slide Time: 08:20)



And then this in this corresponding the processing element block also will be changed initially what was there? Initially it was something like this sum and then carry and then  $X_n$ , but here it is being anded this  $X_n$  is basically anded with  $a_m$  and then it is gone to the corresponding full adder cell and then it is producing  $S_i$  and  $c_{i+1}$  ok

So, this AND gate is basically included in the basic cell and then it gives this particular processing element are copied and it produces the corresponding this array multiplier.

(Refer Slide Time: 09:12)



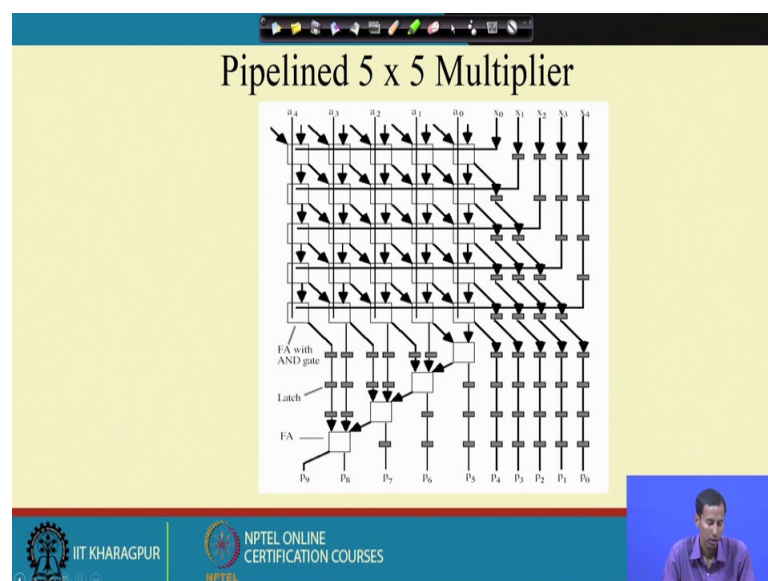
And if I just; that means, put this in an array multiplier architecture so; that means, in array based technique. So, at that time only this  $a_4$ ,  $a_3$ ,  $a_2$ ,  $a_1$  and  $x_0$  and in this

particular this particular direction and in horizontal direction I have to give this  $x_0, x_1, x_2, x_3, x_4$  ok.

And after that I will get the corresponding results  $P_0, P_1, P_2, P_3, P_4, P_5$ ; that means, here I do not have to put; this initially what we are doing? That  $x_0$  and  $x_1$ ; that means, a  $0 \times 0$ ; a  $1 \times 0$ , a  $2 \times 0$  all I am putting at this particular terms, but here I do not have to do that. So, only one  $x_0$ ; that means, the anding operation has been done inside this processing elements ok.

So, this is the unsigned multiplication architecture along with the modified processing blocks. So, modified processing block means initially we are considering only the full adder cell, but here we are considering the anding operation inside the processing block itself ok.

(Refer Slide Time: 10:42)



Then suppose I want to how can I that means, improve the corresponding speed of operation? So, what we know that for array multiplier architecture this is the fixed or that, the delay or the number of stage which I am getting for this the adder stage; they are fixed which is of 8; 8 number of full adder cell.

Now, as that is fixed now how can I suppose; I need to increase the corresponding frequency of that circuit. So, at the time how can I do? If I use pipeline technique at the time I can improvise the speed of operation. So, how I can do that; that means, the pipeline of this array architecture; how can I do? So, as you see; that means, there is total 1, 2, 3, 4, 5 and in this case; so, 5 stage of operation.

So, in the vertical direction what; we give we put this a 0, a 1, a 2; that means, all the a or all the multiplicand bit that we give; the multiplier bit are; that means, applied in the horizontal direction. Now this x 0 that will be first anded or that will come into the first set of this in the first row ok. And that are with no delay element; in the next this x 1 will be delayed by 1 register or it will be delayed by 1 ok. So, then again that will come then x 2 as it has been applied on or the partial products will be generated at the second level; that means sorry third level. So, it has to pass to the 2 delay over here. So, the x 2 has been applied after 2 delays.

So, in the same manner x 3 has to applied after 3 delay and x 4 has to applied after the 4 delay. So, that for each of the stage the delay element I have put; so, this is for the input side; then what will happen for the output case? For the P 0; for the P 0 element it has to pass, it has to pass through the number of delay element. So, actually if I just; that means, so whenever I will apply this a 0 and x 0; it will produce this P 0 terms.

But this for P 9 case how many; that means, what is the number of that register block which I has 2 pass that is basically 4 for this because x 4 has been applied after 4. And whenever I will get; that means, this particular stage if I go back to these. So, at the very end then again I need to pass through this like a carry propagate adder or ripple carry adder for generation of this P 5, P 6, P 7, P 8 and P 9.

So, this can be done based on at this carry is basically propagating. So, initially this P 5 signal that will come for producing the P 6. Then whenever this P 6 will be generated this then the corresponding output from these 2 particular block; they will be passed 2 because that is one level next. So, there two particular signals, they will be latched and they will come after this.

In this particular stage that these 2 will be come after 2 delay element and these 2 will come after 3 delay element; that means, so after 7 delay 4 over here and 3 for this particular case this; P 8 and P 9 will be generated. But if I see this particular P 0 term, so at that time P 0 and x 0 is basically generated at the 0th clock cycle.

This P 8 and P 9 they has been generated on the after the 9 clock; that means, sorry 4 and 3; so, 7 clock cycle, but P 0 x 0 has been produced in the 0th clock cycle because there is no delay element over here. So; that means, this P 0 has to be delayed, so P 8, P 9 that has to be synchronised with P 0; so; that means, this has to also pass through this total sorry 4 over here and 4 for this and 3 over here.

So, after; that means, eighth delay element it has to pass why? Because at the eighth clock cycles only I will get this P 8 and P 9 as this particular cell is producing P 0 at the 0th clock cycle so; that means, this has to pass through 8 corresponding number of stage ok; so P 0 has to pass through this.

Then P 1 has been produced at what? P 1 has been produced at 1st clock cycle so; that means, it has to pass through 7 another delay element to these synchronised with P 8 and P 9. So, that is why if you see; so for P 0 1 2 3 4 5 6 7 8; 8 number of delay element it has to pass. So, after that P 0 will be come over here. So, for P 1 it has to pass through 7 because it is produced at 1st clock cycle; that means, 1 2 3 4 5 6 7.

Then P 2 it has produced at two's clock; that means, second clock cycle ok. So, that is it has to pass through 6; so, 1 2 3 4 5 6; then P 3 that has to at the third clock cycle, it is produced. So, another 5 clock cycle it has to pass; so 1 2 3 4 5; 5 4 P 3, then 4; 4 has to; that means, produced at fourth clock cycle. So, 4 another it has to pass so; that means, P 4 which is passing through 1 2 3 4 ok.

So, then for P what will happen for P 5? P 5 is the fifth, after fifth clock cycle it will produced. So, fifth clock cycle produced means it has to pass through another 3 extra registers; so, 1, 2, 3 ok. So, then again P 6; P 6 is produced after sixth so; that means, now 6 is basically then again it has to pass to the another 2. For P 7 it is at 7 so; that means, it has to pass through another one; P 8 and P 9 that has produced at 8 clock cycle.

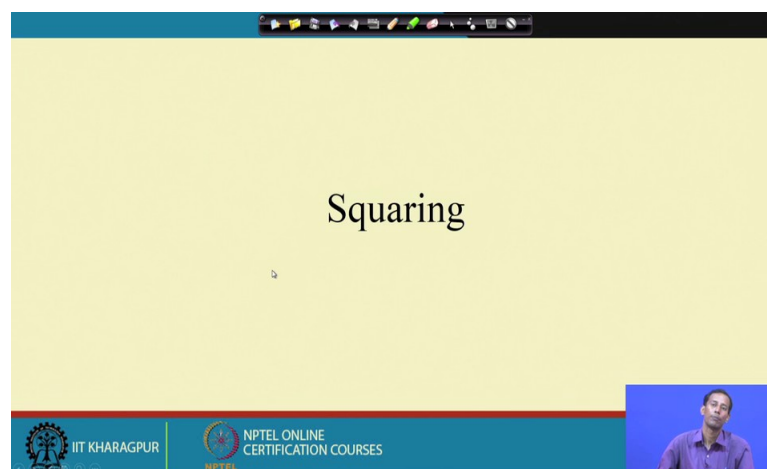


So, this has to pass through no other delay element so; that means, whenever I am getting, at the 8<sup>th</sup> clock cycle I will get this P 0 to P. So, why 8 clock cycles I need or 8 number of register I need? Because 8 number of full adder cell I am doing or adding in this particular array multiplier architecture. So, this is the structure for pipelined 5 cross 5 multiplier ok.

So, now here what is the corresponding delay for this particular case ok? So, now, the delay for this particular case will be of; that means, what will be corresponding delay element over here? Because the inputs are also passing after one delay element; that means, here whenever this will be nothing, but one this processing element blocks delay why is so? Because not this one here only this 4 that processing; sorry 5 total processing element block. So, that will be the corresponding delay over here because at the final level I have to process this through 4 and one process one full adder cell delay for this particular each of this block over here ok.

So, 5 for the final and one for the corresponding delay for this particular cell. So, total 5 I will I have 2 because the carry is basically propagating in this particular delay cell right. So; that means, total 5 delay element full adder cell delay element will be the delay for this pipelined 5 cross 5 multiplier architecture ok. So, this is the pipelined structure of 5 cross 5 array multiplier architecture.

(Refer Slide Time: 21:08)



So, then again; that means, till now we are discussing about this multiplication where this multiplier and multiplicand bit; that means, the numbers there different. Then squaring I think squaring means what? x square or y square or a square or b square means what? I am multiplying the same number; that means, where the multiplier and multiplicand bit or the numbers are same; so at that time that is the square. So, if I multiply a with b at the time it will be a b; otherwise if I multiplying a with a. So, at the time it will be a square ok.

Suppose these 2 multiplier and multiplicand bit they are same; so at that time in regular multiplier architecture if I put that. So, at that time that circuit also will give you the same results, but if I need one squaring circuit at the time as there is 2 the number of multiplier and multiplicand are same; at the time can I produce one optimised circuit for that for the squaring operation, if I can then how I can do that ok?

So, what are the changes will be there or what would be the idea for doing the optimisation whenever we will draw or we will come to the architecture of squaring circuit that we will see now ok.

(Refer Slide Time: 22:55)

The slide, titled "Optimizations for Squaring (1)", illustrates the reduction of a bit matrix for squaring. It shows two stages: "Multiply x by x" and "Reduce the bit matrix".

**Multiply x by x:** A 9-bit multiplier (x<sub>8</sub> to x<sub>0</sub>) and a 9-bit multiplicand (x<sub>8</sub> to x<sub>0</sub>) are shown. The resulting bit matrix has 9 rows and 9 columns. Red arrows indicate that the upper triangular part of the matrix is redundant and can be reduced to zero. A box labeled "Reduce to x 0" is shown next to the upper triangular part. A red circle highlights the diagonal elements, and a red arrow points to the text "Move to x 0" and "Shift column".

**Reduce the bit matrix:** The bit matrix is shown with the upper triangular part reduced to zero. Red arrows indicate the shifting of the remaining bits to their correct positions. A red circle highlights the diagonal elements, and a red arrow points to the text "Move to x 0" and "Shift column".

**Handwritten Annotations:**

- Top left:  $x_2, x_1, x_0 \rightarrow x_5$
- Top right:  $a \cdot a = a^2$ ,  $0 \cdot 0 = 0$ ,  $1 \cdot 1 = 1$ ,  $a \cdot a = a^2$ ,  $a + a = 2a$
- Bottom right:  $x_2 + x_2 = x_3$ ,  $\Rightarrow x_2(1 + 1)$ ,  $\Rightarrow x_2 \cdot 2$

So, in general if I that; that means, I want to multiply x with x where or each of are like 5 bit. So, in this position this will be x 0; x 0 this will be x 0 x 1 and again for this particular case it will be x 0 x 1.

So, x 0 x 0 means what? x 0 is multiplied with x 0 means what this is nothing, but x 0 if I

am a multiplying a with a means that is a why is so? Because if you see if I multiplying a with that is a because if a value is 0. So, 0 into 0 means 0 and 1 into 1 means 1. So, whatever is the value of that I am getting at the output. So, that is why a dot a means that is nothing, but a. So,  $x_0 \cdot x_0$  that will be nothing, but my  $x_0$  ok.

So, then for this particular for p 1 it will be this and this; so  $x_0 \cdot x_1$  and  $x_1 \cdot x_0$ . So, that is nothing, but this  $x_0 \cdot x_1$  and  $x_0 \cdot x_1$ . So, if I that means, add a plus a; so, it will be twice of a; so twice of a means what? The a will come to the next MSB position so; that means, this position, now this  $x_1 \cdot x_0$  that will come to the corresponding next; so, this will move to the next column.

So, now for p 2 what will be the case? For p 2 it will be  $x_0 \cdot x_2$  or  $x_2 \cdot x_0$ ; then it will be  $x_1 \cdot x_1$ ; then it will be  $x_0$  into  $x_2$  ok. So, for p 3 case what will be the case; that means, for one it is this for another one this and for another one this. So,  $x_0 \cdot x_2 \cdot x_1 \cdot x_1$  and  $x_2 \cdot x_0$  ok; so, this and this is same. So, this and this same means; so, this will move to the next column and  $x_1 \cdot x_1$  means nothing, but  $x_1$ . So, this  $x_1 \cdot x_0$  has come to this and it will be added with  $x_1$ .

For this particular position; so now the thing will be that  $x_3 \cdot x_0$  ok. So then  $x_2 \cdot x_1$ , then  $x_1 \cdot x_2$ ,  $x_0 \cdot x_3$ ; so  $x_3 \cdot x_0$  and  $x_3 \cdot x_0$  are same values ok. So, then that will move to the next and here you see  $x_2 \cdot x_1$  and  $x_1 \cdot x_2$  they are also same. So, then again that will also come to the next. So; that means, at this p 3 position what will be there? This  $x_0 \cdot x_2$  only; so,  $x_2 \cdot x_0$  is coming over here.

So, in this manner if I if I just do it for all this position; finally, what we will get? We will get the partial product something like this  $x_1 \cdot x_0$ ;  $x_1 \cdot x_2 \cdot x_0$  at this particular case;  $x_3 \cdot x_0$ ;  $x_2 \cdot x_1 \cdot x_2$  something like this I will get ok. So, then again can I; that means, optimised this more. So, if I add this  $x_2 \cdot x_2$  along with  $x_2 \cdot x_1$  if I add, so at the time it will be  $x_2 \cdot 1$  plus  $x_1$ .


So, what is this value? 1 plus  $x_1$  that is nothing, but 1; so this is  $x_2$  so; that means, only  $x_2$  value I have to add at this particular things. So, the same thing will happen here for  $x_3$ ;  $x_2$  plus  $x_3$  that will be nothing, but my  $x_3$ . So, again more reduction in this; that

means, multiples of this can be done.

(Refer Slide Time: 27:29)

**Optimizations for Squaring (2)**

$\begin{array}{r} x_i x_j \\ x_j x_i \\ \hline x_i x_j \end{array}$	$\begin{aligned} x_i x_j + x_i x_j &= 2 x_i x_j \\ x_i x_i &= x_i \\ x_i x_j + x_i &= 2 x_i x_j - x_i x_j + x_i = \\ &= 2 x_i x_j + x_i (1-x_j) = \\ &= 2 x_i x_j + x_i x_j \end{aligned}$
---	--




So, finally, what I will get? Finally, this can be ok. So, optimisation can be done in this way and finally, for this particular case I will get the terms over here 2; 2, 2 and 2. And then using any first adder if I just add at the time I will get all these values of this.

So; that means, now if I am multiplying; that means, for the squaring circuit; it requires or we have optimised because of this logical; that means, because of this Boolean algebra or this logical optimisation we will get the circuit only considering very nominal number of the basic gates ok. So, that is the squaring circuit; that means the architecture of that squaring circuit; now I will come from this particular equations ok. So, this is the optimisation while we will do the squaring circuit ok.

(Refer Slide Time: 28:45)

**Squaring Using Lookup Tables**

<p>input=a</p> <p>0 1 2 3 4 ...i ...<math>2^k - 1</math></p>	<p>output=a<sup>2</sup></p> <table border="1" style="width: 100%; text-align: center;"> <tr><td>0</td></tr> <tr><td>1</td></tr> <tr><td>4</td></tr> <tr><td>9</td></tr> <tr><td>16</td></tr> <tr><td>...</td></tr> <tr><td>i<sup>2</sup></td></tr> <tr><td>...</td></tr> <tr><td>(2<sup>k</sup>-1)<sup>2</sup></td></tr> </table>	0	1	4	9	16	...	i <sup>2</sup>	...	(2 <sup>k</sup> -1) <sup>2</sup>	<p>for relatively small values k</p> <p>2<sup>k</sup> words 2k-bit each</p>
0											
1											
4											
9											
16											
...											
i <sup>2</sup>											
...											
(2 <sup>k</sup> -1) <sup>2</sup>											



And then using one lookup tables; what is lookup tables? Lookup tables is nothing but your like a; that means, some values which are stored in memory ok. So, this is nothing, but like a ROM structure. So, in ROM what we do? We only put the values and read by giving the proper address ok.

So, in squaring what we know because for 0 it will be 0, for 1 it will be 1, for 2 it will be 4, for 3 it will be 9, for 4 it will be 16. So, we know as the values are same; so we know what will be for 1, 2, 3, 4, 5, 6 something like this. So, what will be the corresponding squaring value; so if I just store it in ROM and then this input that if I just; that means, give as an address to that ROM. So, any; that means; obviously, I will get the corresponding output very easily ok.

So, this circuit will be very much optimised so; that means, I do not have to use any extra logic gates or logical terms to produce the corresponding squaring of the numbers. But this is advantageous when the number of; that means, the length of input that I am considering lesser because for 4 bit there will be 16 number I will get so; that means, in the ROM table the depth of ROM will be 16.

If I consider this a of 10 bit. So, at the time what will be the number of combination over here? So, that will be  $1024$ . So, at the time it will; that means, produced this; that means, it will request the depth of this ROM; that means, depth of this ROM that is  $1024$  not only this  $1024$  depth it will increase this width of these too ok.

So; that means,  $1024$  number along with the corresponding depth; that means, that is of 20 bit for each of this storing this particular the; that means, this one particular ROM bit. So, I require 20 bit so; that means, total  $1024$  into 20; so, that will be the size of this ROM. So, at that time it will consumes more power and it will be slow in operation in compared to if we implement those squaring circuit using logic gates. So, at the time this lookup table base squaring circuit is not that much efficient way to implement it.

So, for the lower number of values or the lower width we consider for a at the time it is advantageous this lookup table base, otherwise this structure is not that much advantageous in terms of power or in terms of speed or in terms of area ok. So, this is

the; that means, another squaring technique or squaring architecture for how we can implement or how we can do. And what I suggest that it is whatever we are discussing here that is not the end. People are basically day by day they are working or they are doing research on this different architecture, finding out different architecture for high performance multiplier architecture ok.

So, more if you are interested on more on this learning on this particular multiplier architecture please follow any scientific websites like IEEE or Google, Google scholar ok; so, any of this and use your; that means, and from that particular point you come down how you can; that means, you note down what are the problems or why that or how you can improvise the corresponding circuit and then you try to build your own multiplier architecture for your particular application ok. So, this is; that means the  $n$  for integer multiplication ok.

So, now in the next we will see this suppose in some of the case; I need to multiply floating point numbers. So, at the time how can how can I do? Though we are in the fixed point domain; so, how can I do this floating point number multiplication ok. And what are the; that means, way of optimisation or is there any way of optimisation that we can do whenever we will consider this; so that you will see from the next class onwards ok. So, this is it.

Thank you for today's.