

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

**Lecture – 31**  
**Multiplier Architecture (Contd.)**

Hello everyone. So, welcome back to the course on Architectural Design of ICs. So, we are seeing different multiplier architecture so, in the last class we have seen Booths algorithm ok. So, using Booths algorithm why we use Booths algorithm because in normal algorithm the partial product generation so, that we basically shift and add ok. Using adder tree reduction method, we can reduce the depth of that; that means, the addition operation why I require that addition operation in a chain, because I have to add those partial product generator; that means, one by one the which 1 are coming by 1 bit shift ok.

So, to reduce those addition operations in chain, we introduce these Booth algorithm and from that example we have seen that, using Booths algorithm we can if you use radix 4 at the time we can reduce that partial product term by 2 ok. So, and if; that means, we also can; that means, consider radix 4 radix 16. So, more on this radix I can increase.

So, after that what we have seen that, if I use any negative number so, at that time what I have to do for a negative number I have to extend the sign ok. So, otherwise what will happen if I do not use; that means, if I do not use this sign extension method for this Booths multiplication, while we are following this Booths algorithm at the time what happens? We will get one erroneous results ok,

So, to get proper result we have to extend the sign to the corresponding MSB side that in the example we have already seen, but whenever to get the proper result. So, what actually what is our intention for using Booths algorithm? To reduce the corresponding number of addition operation which is required for adding the partial products, right?

Whenever we are; that means, extending those sign bits into the MSB position. So, at that time we are basically increasing the number of full adder cell requirement; why because that position is filled. So, initially that position was blank, but now as I have extended those or though I have filled those positions with this sign extended sign bits

so; that means, I am increasing the number of full adder cell.

So, to deduce those full adder cells what is there means, I can do one method; that means, which is known as this template method ok. So using template method I can reduce those number of sign bits, and I can also get the benefit of; that means, Booths algorithm at that time in a better way or to design one high performance multiplier circuit ok. So, how to do that? That we will see in todays class.

(Refer Slide Time: 03:48)

**Multiplication Output Range**

- Multiplication:  $A \times B = C$ ,  $A = K.L$  number,  $B = M.N$  number
- Unsigned multiplication
  - $K.L \times M.N = (K+M).(L+N)$  number
  - $1.3 \times 1.4 = 2.7$  number
  - Example:
    - Binary:  $1.101 \times 1.1001 = 10.1000101$
    - Decimal:  $1.625 \times 1.5625 = 2.5390625$
- Signed multiplication (two's complement)
  - $K.L \times M.N = (K+M).(L+N)$  number
  - $1.3 \times 1.3 = 2.6$  number
  - Example:
    - Binary:  $1.000 \times 1.000 = 01.000000$
    - Decimal:  $-1 \times -1 = 1$
    - Binary:  $0.111 \times 0.111 = 00.110001$
    - Decimal:  $0.875 \times 0.875 = 0.765625$
    - Binary:  $1.000 \times 0.111 = 11.001000$
    - Decimal:  $-1 \times -0.875 = -0.875$
  - NOTE: Only need  $K+M$  integer bits when  $A$  and  $B$  are their **most negative allowable values**
    - If  $A$  and  $B$  can be restricted such that they do not reach most negative allowable values, then only need  $K+M-1$  integer bits, i.e. output results is  $(K+M-1).(L+N)$
    - This is a useful trick often using in DSP systems to reduce wordlengths!

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, basically this is the multiplication; that means, output range. So, output range means why I have to define this. Because, suppose I have to multiply  $A$  with  $B$  and  $A$  is of number of; that means, the bit width or the wordlength of  $A$  is  $K$  dot  $L$ , and for  $B$  is  $M$  dot  $N$ . So,  $K$  dot  $L$  means  $K$  is the wordlength which is required for the integer part and  $L$  is the; that means, bit width or the wordlength which is required for the fractional part the same thing happens for  $B$  where  $M$  for integer  $N$  for the fractional part.

So, in actually there are 2 types for multiplication; one unsigned multiplication and one sign multiplication. So, till now whatever we are following that is unsigned multiplication. So, later on this course we will discuss on sign multiplication method. So, if I multiply with this  $A$  and  $B$  which are of; that means,  $K$   $L$  bit and  $M$   $N$  bit. So, at the

time the final product C will be of K plus M dot L plus N number; that means, wordlength for c will be K plus M and L plus n.

The same things happen for the sign multiplication too. So, if we show this example or if you see this look into these particular examples which are mentioned here; that means, for the binary if I use that 1.101 dot 1.1001. So, at that time C will be 1 0 and the corresponding; that means, the fractional part will be 7 bit in a length ok.

So, because here we have considered 3 here we have considered 4. So, total 3 plus 4 7 for the fractional part, you here you see 7 bit for factional part and 1 bit here 1 bit here so; that means, for the integer part I need 2 so; that means, there is 2 bit. So, for the say decimal also the same thing; so for signed multiplication also it; that means, follows the same rule.

(Refer Slide Time: 05:55)

### Comparison of Booth and parallel multiplier shift and Add

<pre> Multiplicand 010101 Multiplier   001010 ----- 000000 010101 000000 010101 000000 000000 000000 000000 ----- 000011010010 Result           </pre> <p style="text-align: center;">REGULAR SHIFT AND ADD MULTIPLICATION</p>	<pre> Multiplicand 010101 Multiplier   001010 ----- 111 ----- 00000001010 0000001010 00001010 ----- 000011010010 Result           </pre> <p style="text-align: center;">BOOTH MULTIPLICATION</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

IIT KHARAGPUR

NPTEL ONLINE  
CERTIFICATION COURSES

So, this is the comparison of Booth and parallel multipliers shift and add method. So, in shift and regular shift and add multiplication method what we do? Basically if I this is my multiplicand and this is my multiplier then in each of this stage, this these are the partial products which I have been generated. So, as I am doing six bit multiplication; that means the multiplier with this 6 bit so; that means, 6 set of partial products I have to

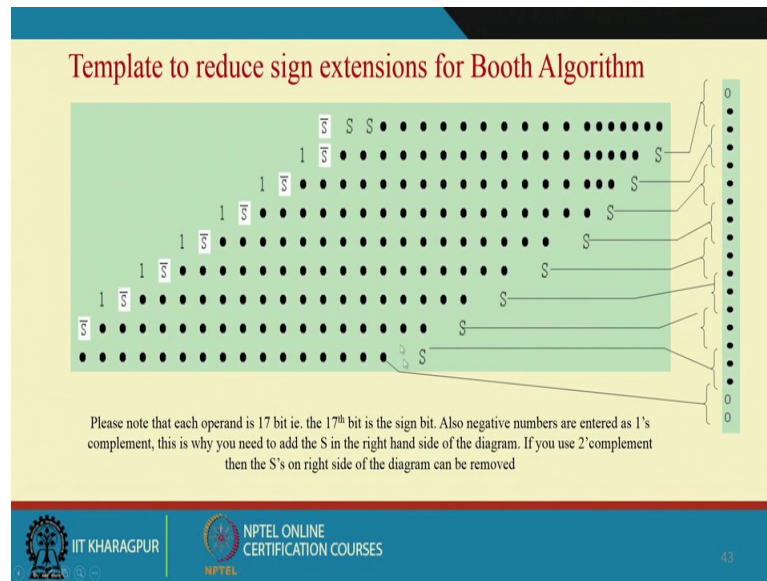
generate and then finally, I have to add those partial products ok. But in case of that radix 4 Booths multiplication what we have to do? We can recode or we can encode this multiplier bit considering 3 bit together.

So, whenever we are considering 3 bit together for this particular six numbers, I can get 3 combinations ok. So, for 3 combination I will get 3 bit shift; that means, 3 bit partial products and as I have consider radix 4; that means, that is why there will be 2 bits shift here you see here there is only 1 bit shift, but here there is 2 bit shift 2 bit left shift. If I chose radix 4 at that time this will be 4 bit shift so; that means, now this if I want to add these partial; that means, six partial products. So, at the time I require 5 full adders in general. But here to; that means, to add these particular partial products, which are being generated after using Booth multiplication; that means Booth algorithm.

So, using 2 adders only I can get the responding results which are same as this. So; that means, I can reduce the number of orders in a great way or in a better way. So that means, the performance of this particular circuit or this particular multiplier that will be much better than this one the general shift and add multiplication method. So, that is why we follow this Booth multiplier ok.

So, there are; that means, several researches are going on to find out this the optimised or the; that means, the best circuit for this multiplier design using Booths. So, many of the; that means literatures are available on Google or I EEE site. So, if you are interested or if you are working on this particular multiplier architecture; so you can inform me via; that means, this discussion forum or you can search it in Google or I EEE site or any scientific; that means website.

(Refer Slide Time: 08:44)



So, then what I said, to reduce the number of; that means, signs which I have been extended for this Booth multiplication, that we can reduce by using the template method ok. So, according to the template method what it says that, so, this is the multiplier bit at this right hand side ok. So, for the first hand for the; that means, first partial products, this will be as I have; that means padded 0 at the LSB position ok. So, I have to do that in any case; whenever we are considering this 3 bit combination. So, at the time I have to add 0 at the LSB position and then I will start to combine those 3 bits ok.

So, for the very first row according to this template method and this in the very first row will be; that means, the MSB 2 position, here that will be sign and sign and then it will be this sign bar. So that means, these 3 positions whatever this is the corresponding multiplicand bit or this is the that means each bit corresponds to the multiplicand bit. Multiplicand bit means based on the corresponding whatever is the result after this decoding ok.

So, but for these 3 MSB position, the sign will be extended like this; that means, it will be; that means, for this encoding if I get that that is 0 in Booths multiplication what we got? Then; that means, there the combination will be 0 a or minus a or 2 a or minus 2 a correct so; that means, for that reason. So, for 0 or for a or for 2 a the sign bit is 0 as the

those are positive; that means, positive number. For minus a and for minus 2 a the numbers are negative. So, at the time sign will be 1. So, that is why those sign will be extended to this particular position for the very first row or the very first partial product; that means, for the very first combination of this.

So, after that after whatever is the; that means, the combination I will get, that for that reason these will be the MSB will be 1. And next to the MSB that will be the S bar and that we will follow for the rest of the; that means, this partial products ok. And here you see whatever sign I am getting so, that is being added to the LSB position ok.

So; that means, whenever for the first row whatever sign I am getting, that has been added to the LSB position. For this case this is added to the LSB position of those partial products. So, why I am doing that because here what I am doing based on the sign, we are basically calculating whether this number is positive or negative. If this number is positive, at the time I am adding 0 to the LSB means I am doing nothing and here also this is 0 means nothing, but if I am getting 1 over here. That means, the negative number at that time I am just the partial product making the partial product just twos complement of that by inversion here and then addition of 1 over here at the LSB position. So, that is the twos complement method right.

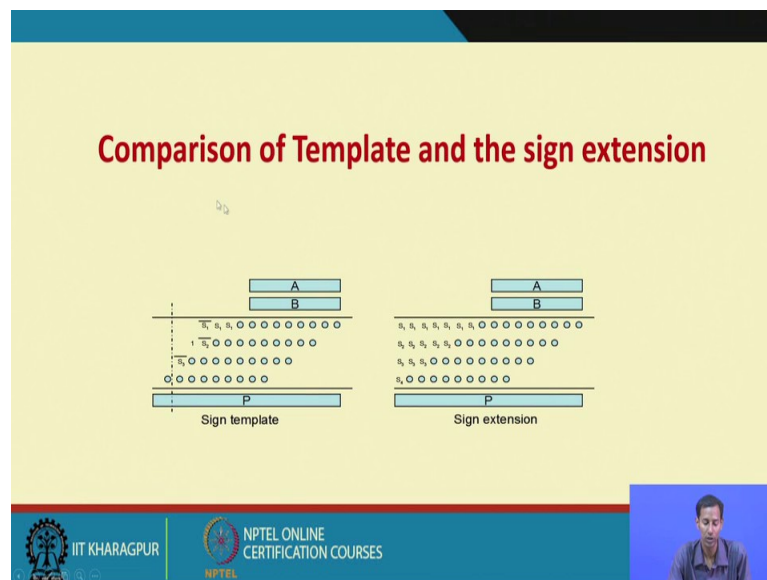
So, the same thing is basically happened to all of this so; that means, for that reason we are doing that and up to what position we are doing? Up to the last partial; that means, before to the last the partial products which I want to generate that will be only S bar not 1 and S bar. So, after that after that the last partial products there I will not use any of this why I will not use any of this? Because this is my; that means, this 17th bit position and here I do not have to extend the sign from this particular bit because this is already this bit is already coming to the MSB of this.

So, that is why here also; that means, the last 2 partial products is not considering the sign extension as 1 and S bar. For the last as this MSBs coming over here so, that is why there is no such sign extension and before to that, only one position is left so that is filled with S bar. But here you see for each of these the sign extension I have to add the sign at the LSB position which I am doing in this particular right hand side ok.

So, after doing this you will get the; that means, after now this is for 16 bit Booths multiplication 16 into 16. So, that is why there are total 17 bit ok. So, this 17 bit and here you please note that the operand in 17 bit that is the sign bit and also negative numbers are entered as ones complement. And this is why you need to add the sign in the right hand side of the diagram that is why and if you use twos complement, then the S on the right side of the diagram can be removed.

That means, if we if you use ones complement at the time and in this particular case, I have used that one's complement. So, that is why the sign has been added at the LSB position to make it the twos complement number. So, if you are from the beginning if you are using twos complement of this. So, at the time you do not have to use this S you just remove this S. So, automatically this number is already becoming twos complement itself. So, then we do not need to add this as only the sign extension like this will be there.

(Refer Slide Time: 16:08)



So, this is that means, the comparison of this sign extension method is that, in the sign extension what we do? The sign S 1 has been extended something like this, S 2 has been extended something like this S 3 and then S 4, but here are what we are doing? S 1 is extended only 3 bit position S 2 has been increases only 2 bit 2 bit position and S 3 for 1

bit position and S 4 for nothing.

So, that is why here you see the number of full adder cell which I require for this particular to calculate because of this extra overhead of sign extension that can be removed here in a greater way so that means I can use lesser number of full adder cell in compared to this sign extension method.

Whereas, where whenever we will do optimise at the time the main thing is that, the functionality will remain same all the time that is my; that means, like a main mantra so that means, here the functionality will remain same, but I can optimise or I can reduce the number of full adder cell by the using this sign extension or this template method.




(Refer Slide Time: 17:22)

### Example of using the template

**Using the Template 25 \* -35**  
 $25 * -35$  with -35 as the multiplier. Using 8 bit representation

	Sign bit	
	00011001	
	110111010	
	-----	
Add SS	10000011001	* 1
Add inverted S	1011100111	* -1
Add Inverted sign and add 1	100110010	* 2
Add Inverted sign bit	1100111	* -1
No sign bit	-----	
	11110010010101	
This is a -ve number. Convert it		
	00001101101011	

512 256 64 32 8 2 1 = 875

So, now if you take one example of 25 into minus 35 sorry 25 multiplied with minus 35. So, how we can do that? So, 25 can be represented as 1 1 0 0 1 and I have to use this 8 bit representation. So, 8 bit representation of 25 is this. So, minus 35 can be represented using of 8 bit something like this ok.

So, then what I have to do for this Booths encoding what I can I have to do? I have to add 0 at the LSB position. Now what I will; that means, use that is that for this 0 1 0 or 0



1 0 these 3 combination I will take first. So, for 0 1 0 what is that; that means, the number? That will be a so; that means multiplicand multiplied with 1. So, multiplicand multiplied with 1 if I, then at the time what will be? All these position will be filled and if I just go back to this and then this I have to extend the sign for 3 bit position right.

So, 3 bit position how it will be extended? 2 for whatever is the sum sorry sign and the MSB that is sign bar so; that means,. So, sign bit here this is positive number; that means, positive one. So, that is why this is sign is 0 0 and sign bar is 1. Then for this 1 1 0 for 1 1 0 what will be the combination what will be the; that means, from the decoding take will what will be the output? That will be minus 1

So, minus 1 means what? Minus 1 means the corresponding complement of this particular circuit twos complement of this circuit ok. So, twos complement of the circuit if we chose so, at the time what? This will be 1 this will be 1 1 0 0 all 1. So, 1 1 1 0 0 all 1 and then how the sign will be extended for this case? 1 and S bar. So, S bar in this particular case S bar is 1 and this is negative number so; that means, S 1 and S bar is 0 and here you see I have not put extra S to the LSB why? Because here I am already using twos complement for minus 1, I have already use a twos complement. So, that is why I do not have to add S.

So, if I use ones complement at that time only for this particular case at that time only I have to add this S otherwise I do not have to. So, then for 0 1 1 case what is the; that means, decoding combination? That is positive 2. So, for positive 2 what I said? Because that is the last to the; that means, this is the just the before to the last partial products ok. So, because 4 combination I will get. So, this is the third. So, for the third or for the last to the before to the last that means what is the sign extension a sign extended things that are only S bar ok.

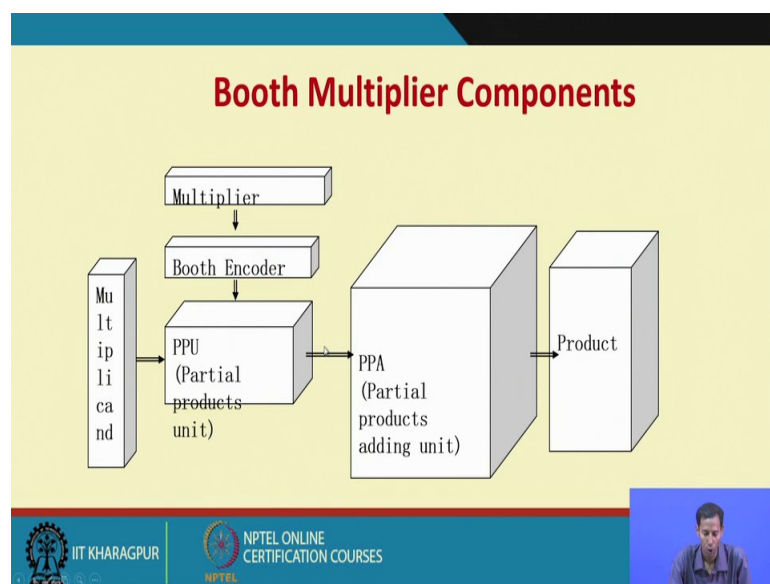
So, here only 1 bit will be extended oh sorry; that means, in extended using template method, that is S bar in this case sign is positive. So, S bar will be 1 over a. So, and at the very end what will be the case? That means, for this is 1 1 0 ok. So, 1 one 0 means that is again minus 1. So, minus 1 means just the inversion of that that means twos complement of that. So, twos complement of; that means, 1 1 1 0 0 then 1 1. So, you can; that means,

add this extra 1 or you can remove it because I need 8 cross 8 means I need total 16 in number over here.

So, after adding this each of this position now if you just add, at that time what will happen? If you just add then at the time what will happen? Then finally, you will get all 1 over the end and then this. So, this as this bit is 1 so; that means, this indicates the results is negative number. So, negative numbers means the results magnitude how we I can find out? I can find out if I took twos complement of that ok.

So; that means, here I need to take the twos complement of these which if I chose this twos complement of that, then I will get the magnitude of that result which are 1 then 1 0 for 1 for 1 it will be 0 for 0 it will be 1 for 0 it will be 1 or 1 it will be 0 1 1 then 0. If I find out the; that means, corresponding decimal value for this binary. So, it gives me 875. So, the results are sorry the result is minus 875. So, if i; that means, if I what was my requirement? I have to multiply 25 with 35 which is nothing, but minus 875. So, here also I am getting minus 875 using the sign extension method. So, you can consider any number any number and by this method now we can try whether this algorithm is following or not.

(Refer Slide Time: 24:06)



So, then what will be the architecture for this the Booth multiplication ok. So, the multiplier architecture is here we; that means, showing the corresponding block diagram level representation of Booths multiplier, here in these 2 particular directions, I am having this multiplier in this particular direction and multiplicand in this horizontal direction.

So, the multiplier will be passes through the Booths encoder block why because I need to generate the corresponding combination. So, while we are choosing this the; that means, the 3 bit together. So, at the time what is the outcome; that means, the decoding from the decoding; that means, the from the Booths table what is output for that those combination that we will get from this Booth encoder part.

So, after getting that so, the multiplicand you need that will be that will be minus a or 2 a or minus 2 a or; that means, a. So, based on that; that means, whatever is the multiplicand bit that will be inverted or not inverted or directly it will come or that will be shifted by; that means, for 2 a whether that will be shifted by 1 bit. So, that basically based on this particular decoding; that means, results, that will be chosen in this partial product unit this is the generation unit; that means, from this I am generating those partial products.

After that I need to add these partial products and finally, I will get the products. So, whenever we will use this partial product adding unit so, at that time I can use this 3; that means, 3 kind of things which we have already seen this Wallace tree method. So, tree reduction method I can use for this partial product generation, which I am doing after getting the partial products generated by using Booth multiplication Booths algorithm. So, this is that means, the architecture for Booths multiplier.

So, here also the same thing whenever we are generating the products, at that time 1 thing we will be remember what is that, so at the very beginning what we have seen? If the sign bit is 1 then I have to choose twos complement of the final results; if this bit is 0 then it will directly pass; that means, whenever we will design at the time to choose the; that means, to get the correct results over here. So, there will be 1 multiplexer where this output will be passes through in one path that is through twos complement and in one

path it is it will pass 2 directly to get the proper results; why and what will be the select line? Select line will be MSB of the final results which I am getting after this partial product generation unit.

So, this sign bit for the MSB which is nothing, but the MSB that will decide whether that number will be twos complemented or it will pass directly. So, 0 means, it will directly pass to the output or the output is positive number if it is 1 so; that means, the number is negative ok. So, this is the Booths multiplier architecture ok.

. So, again in the next few classes we will see this different kind of tree multiplication and why I have to use tree multiplication, then we have discussed signed multiplication sorry unsigned multiplication till now. So that means, how to do that signed multiplication and what are the; that means, problems or what are the challenges comes whenever we consider this signed multiplication. So, in unsigned multiplication we can do because this is positive number so, I do not have to consider any other things. But if I use signed multiplication, so at that time what will be the problems or how you can tackle those problems that we will see later on to this particular multiplier architecture topics ok.

Thank you for today.