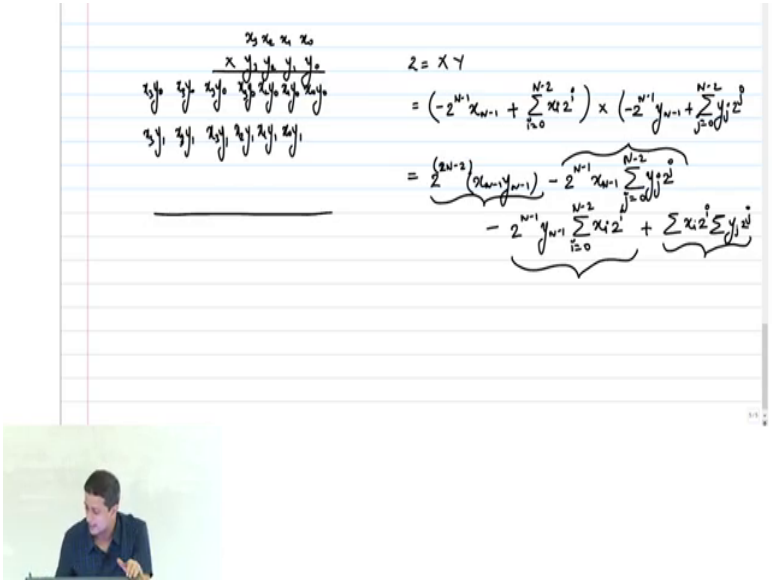
Digital IC Design
Prof. Janakiraman Viraraghavan
Department of Electrical Engineering
Indian Institute of Technology, Madras

Lecture – 65
Carry Save Multiplier- Signed Multiplication

(Refer Slide Time: 00:14)



So, now let us see what we have to do. Suppose, I take this x 1 x naught y 3 y 2 y 1 y naught and I tell you both these numbers are 4 bit twos complement numbers x naught y naught x 1 y naught x 2 y naught x 3 y naught right, then I shift a another. I will just write one more row x naught y 1 x 1 y 1 x 2 y 1 x 3 y 1, what is this term here?

What is this term here? It is x 3 y naught, because I have to sign extend all these partial products right. So, this also has to be x 3 y naught, this is x 3 y naught x 3 y naught. Similarly, this I have to do x 3 y 1 x 3 y 1 and so on, very evidently I need to throw in so

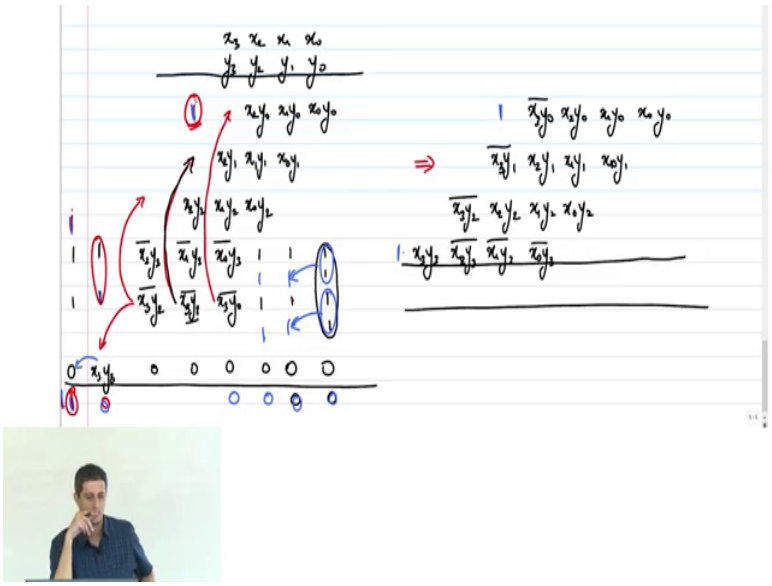
many more adders, if I have to do it this way and therefore, let us not even go down that highly, it is a waste of time and if not I am going to be efficient either in speed or in performance, I mean speed or in area ok.

So, let us see how to overcome this problem in a more intelligent way. I want to do Z equal to X into Y ok. What is this? This is nothing, but minus 2 power N minus 1 x N minus 1 plus summation i equal to 0 to N minus 2 x i 2 power i multiplied by minus 2 power. Let us assume now, both are N bit numbers though, because the focus is not on N and M. Now, y N minus 1 plus summation y j 2 power j j equals 0 to N minus 2.

What I am going to do is I am going to write out this negative term multiplication separately from this positive term multiplication ok. So, this first term minus 2 power N minus 1 into minus 2 power N that will vanish right, you will get 2 power what? 2 N minus 2 into x N minus 1 y N minus 1 minus 2 power N minus 1 x N minus 1 summation j equals 0 to N minus 2 y j sorry y j 2 power j, then third term minus 2 power N minus 1 y N minus 1 summation i equal to 0 to N minus 2 x i 2 power i plus the positive term multiplication. Now, which is summation x i 2 power i summation y j 2 power j.

Now, this is basically my usual multiplication unsigned multiplication and therefore, I can go ahead and just do it normally. This also is a positive number therefore, I have nothing, no problem, only place that I have to worry about are basically this negative term and this negative term right. Wherever you get the highest msb of either x or y right and the other term is not the msb, then it is a negative number and therefore, I have to handle that slightly differently.

(Refer Slide Time: 04:57)



So, what we do is we will do exactly that in the following multiplication x 3 sorry, x 2 x 1 x naught y 3 y 2 y 1 y naught right. So, I will just write this x naught y naught x 1 y naught x 2 y naught, positive numbers no problem similarly, x naught y 1 x 1 y 1 x 2 y 1. Here, again this is what x naught y 2 x 1 y 2 x 2 y 2.

Now, I will not write the other terms, because it the moment it involves x 3 and y 3, I have to be careful. So, we will see what that is, if I took this y 3 term normally, it could have been x naught y 3 x 1 y 3 x 2 y 3, we leave the x 3 y 3 out, because that is again a positive number right. This is now a negative number.

How do I make a number negative? You do, you have to take twos complement of that number which means you invert them and add 1 to it. So, I will invert these numbers of course, these are 0s earlier I am going to now make this 1 right. Similarly, there is this x 3 y naught x 3 y 1 x 3 y 2 term which will come here again right.

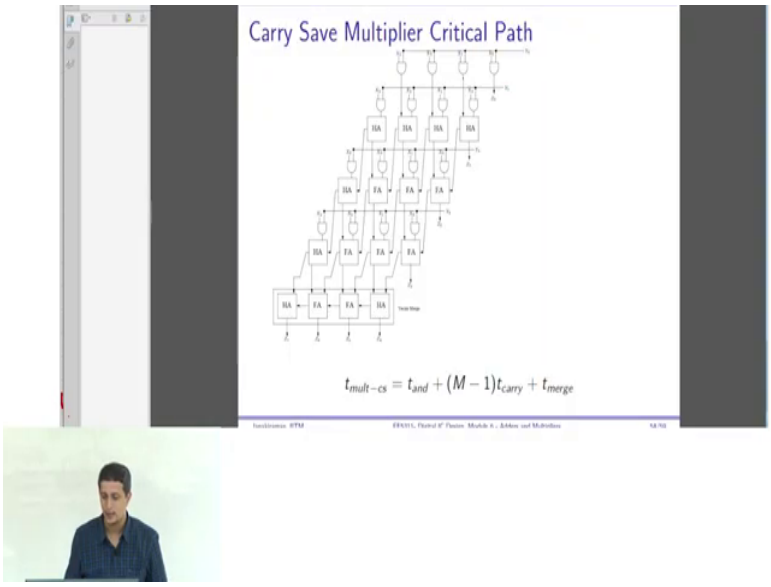
This is x 3 y naught x 3 y y 1 and x 3 y 2, that is also a negative number. I will invert that also this is 1 1 1. I am not putting the ones on the left yet, because we do not know how many bits we need, we will come to that and then what should I do I need to add a 1, because you invert every bit and then add a 1 that is how you get the twos complement number right. That is what I am going to do here right.

Now, I have this x 3 y 3 which is a positive number, no problem right. So, I need to put one more 1 here and of course, I will get a carry therefore, I am going to append two more ones on this side. This is 0 and all these bits are 0 ok. All we have done is taken the negative numbers separately and written it in twos complement form complemented every bit and added 1 to it.

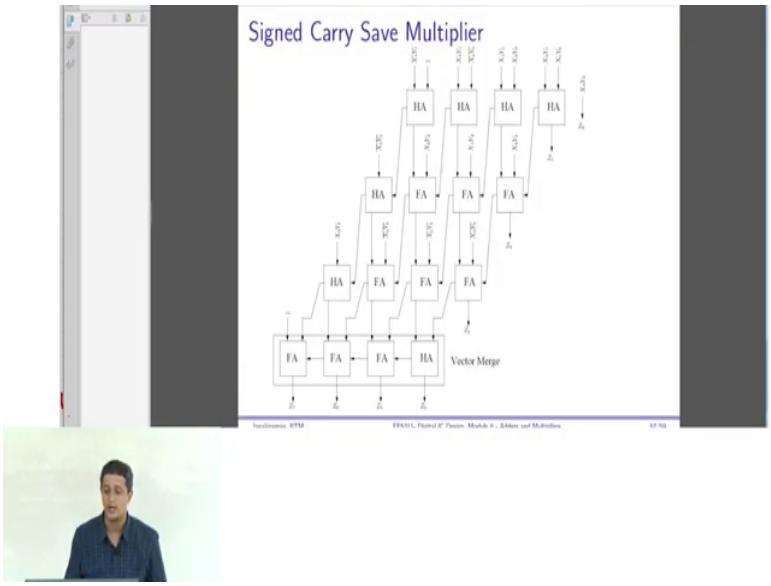
Now, there are a bunch of hard coded ones. You perform the addition beforehand, I have to add four ones what will I get 0 and a 1 will carry over here. Two ones 0 all right, 0 and a 1 will carry over where to 2 bits away. So, you will get a 1 here. So, that is not 2 bits, this is 2 bits correct, 1 2 3 4 ok. If you add these two 1s you will get a carry 1, four 1s if I add I will get 0 0 and a 1 there right.

Now, similarly you come here that addition is d1, I have some 1 dangling, somewhere I have to take care of that in my multiplier add these two 1s to get a 0 and then you get a 1 here. So, you get what 1 1. Now, you simply rewrite this by wherever there was x naught y 3, x 1 y 3 or x 2 y 3 you simply complement it right and you got to add this 1 and you got to add this particular 1 into my adder there right.

(Refer Slide Time: 10:06)



(Refer Slide Time: 10:10)



Therefore, you will find that in order for me to do sign multiplication all I have to do is make this x 3 y 1 wherever there is x 3 y j where j is between 0 and 2 complement the bit or wherever there is y 3 x i and i is between 0 and 2 complement the bit that is what you see here, in that half adder that is why I said retain that half adder, because when I go to sign multiplier I have to put now that 1 here right in my previous thing I showed you that there is the 1 here right along with this x 1 y 3 or x 3 y 1 complement.

I need to add this 1 and that is what is appearing here along with x 3 y 1 bar I have added that 1 to my half adder and therefore, that should remain now that will cascade correctly to the next state next stage finally, I have to add this 1 where did that 1 come from, because I add these two 1s I get 0 and then I add a 1 on top there.

So, the final carry out generation after adding this x 3 y 3 with this. So, this will give me a carry right. This addition will give me x 2 y 3 x 3 y 2 will give me a carry that carry along

with the x 3 y 3 when I add right x 3 y 3 along with the carry coming from the previous stage that will give me a carry and I need to add the 1 to that, is that clear.

So, last part did you follow this arithmetic here, why I could sure. So, let me write this a little differently now ok. I am going to write this a little differently where x 3 y naught I will put it back here, x 3 y 1 bar I will put it here and x 3 y 2 bar I will put it here ok.

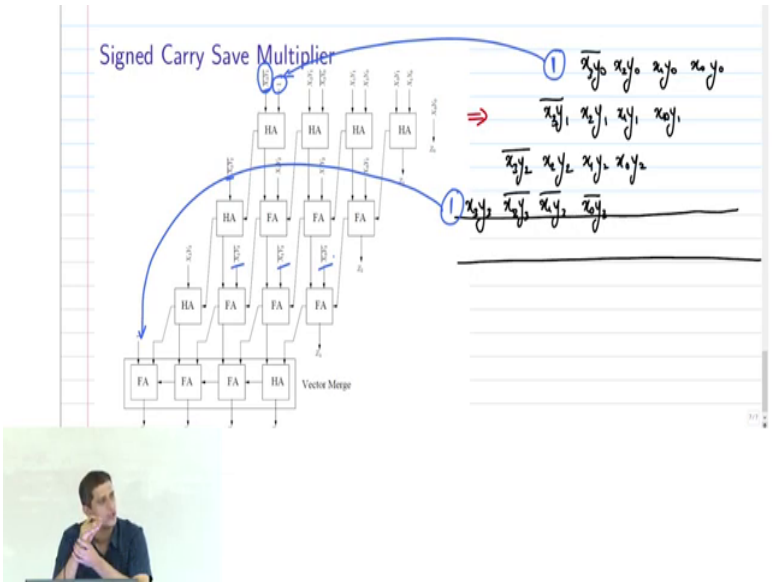
So, this is like x naught y naught x 1 y naught x 2 y naught x 3 y naught bar, then I have x naught y 1 x sorry x x 1 y 1 x 2 y 2 and x 3 sorry, this is y 1 y 1 bar right, that is basically, which term that we got we got this x 3 y 1 bar here, that is what I am writing. I am just rewriting that same addition there right, then what is my third term partial product, it is x naught y 2 x 1 y 2 x 2 y 2 and x 3 y 2 bar right.

Now, there is the another term where I am now going to multiply by y 3 y 3, which is all negative numbers. So, therefore here, I will write this as x naught y 3 bar x 1 y 3 bar x 2 y 3 bar and x 3 y 3 ok. Now, in the process of adding these four 1s, what did I get if I add these two 1s, first of all I will get a carry of 1 add these two 1, I will get a carry of 1. So, the answer is 0 here. So, nothing to be done on that row on that column. Now, again there are four 1s I will get 1 more carry of 1 here carry of 1 here 0 here.

Now, four 1s when I add I will get 0 0 and I have to add a 1 here. So, along with my x 3 y 1 or x 1 y 3 and x 2 y 2 I need to add a 1 right, that 1 I am putting here that addition is done, those ones, because of the twos complement addition we are done, we have ones. Now, on the msb side right. So, therefore, I add these two 1s what do I get 0 and I will get an extra 1 here.

So, if I do 1 plus 1 plus 1, I will get 1 1 right. So, I need to add one more 1 there right, because finally, this 1 needs to go somewhere. Where should it go? It needs to get added to the carry generated from here. So, along with x 3 y 3 I will add a 1 right. It is not along with that actually, it is here, right that is what we are doing here. Now, let me just copy this right.

(Refer Slide Time: 16:37)



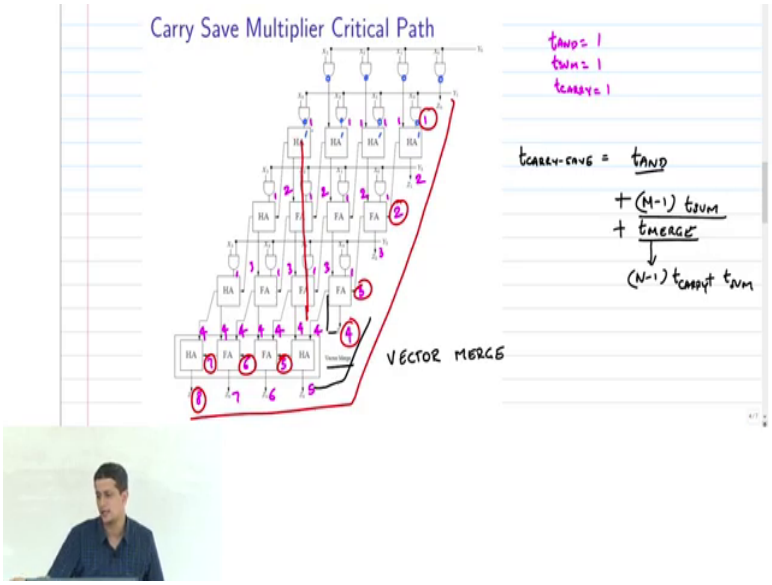
So, we had two 1s that we had to add, this sorry, this 1 needed to be added somewhere where should it be added? It has to be added where you are adding x 3 y 1 x 2 y 2 or x 1 y 3 that is why where we are doing x 3 y 1 up to that has to become x 3 y 1 bar, I am going to add that 1, this 1 comes and sits here.

After doing addition with x 3 y 3 there is a carry out there, I need to add this 1. This guy is now coming and sitting here. So, all I had to do to my unsigned multiplier was just complement the bits where I had x 3 y j, where j was between 0 and 2 just make it complement bits right.

This bit, this bit and not that 1 right, then wherever I had y 3 x I where I was between 0 and 2 again, this 1, this 1 and this 1 I made it complement right and then I had to add 2 1s and that is what I have d1 here, it is quite surprising that it was a simple change like this it is very

counter intuitive unless, you know what the Math is ok. Otherwise, you are never be able to figure out you know how this gives me signed multiplication clear, any questions. Final thing, I am going to leave as you know an optimization for you in the project, all through we have done.

(Refer Slide Time: 18:38)

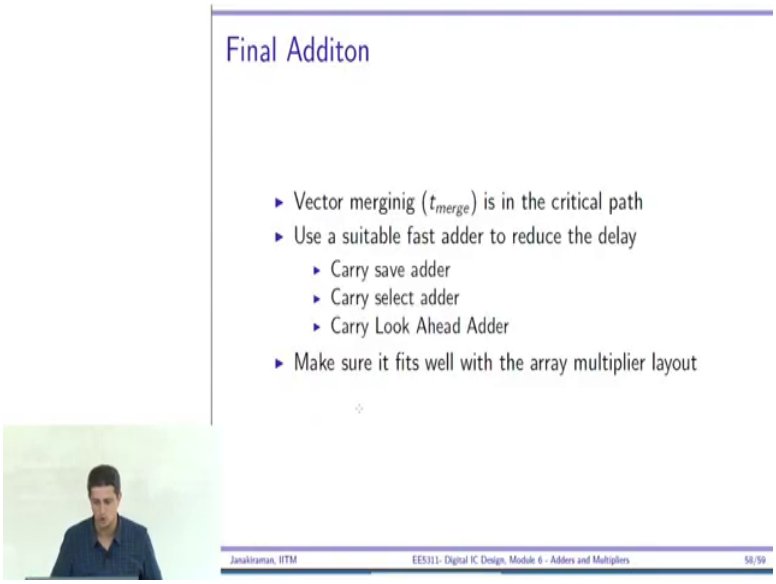


And gates, but of course, since this is in the critical path it is better I do NAND gates, if I make everything NAND, then I need to change, this it has to become a half adder prime in the sense where it takes complements and gives out complements. So, when you design your multiplier please, make sure you optimize for this NAND gate as opposed to trying to use an AND gate ok.

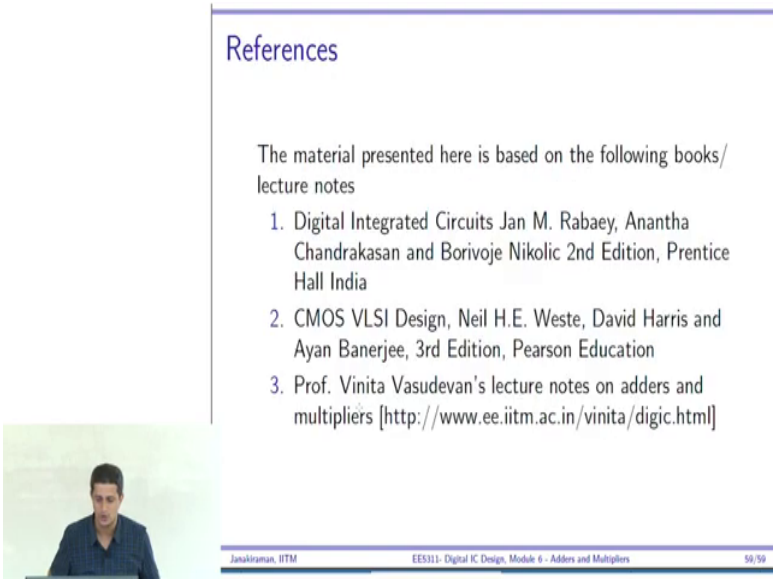
You have to just figure out how to put this you put NANDS, then you had to you will have to have a full adder, which gives you know the complement of sum and carry, then propagate to

the appropriate way ok, with that we have sort of really covered you know the key aspects of multipliers both signed and unsigned. So, booth as I told you, I do not have time. So, I will not do it right.

(Refer Slide Time: 19:45)

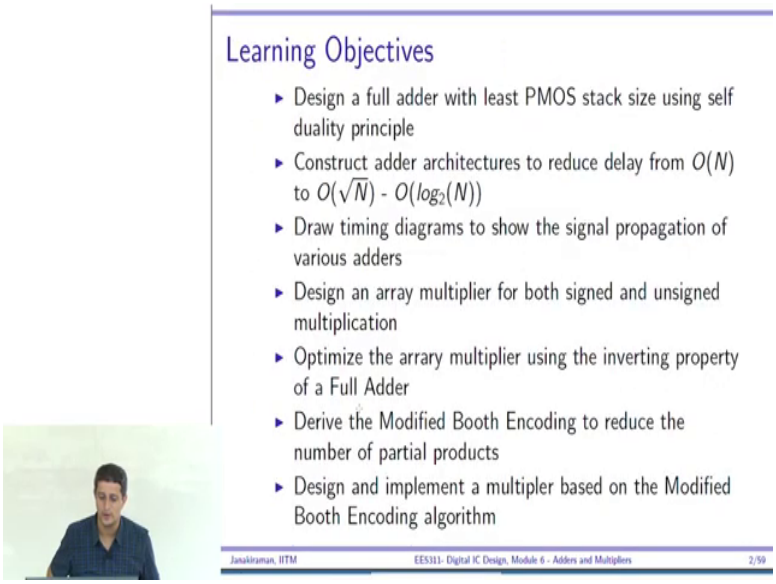


(Refer Slide Time: 19:49)



So, let me just quickly summarize again what we did.

(Refer Slide Time: 19:53)

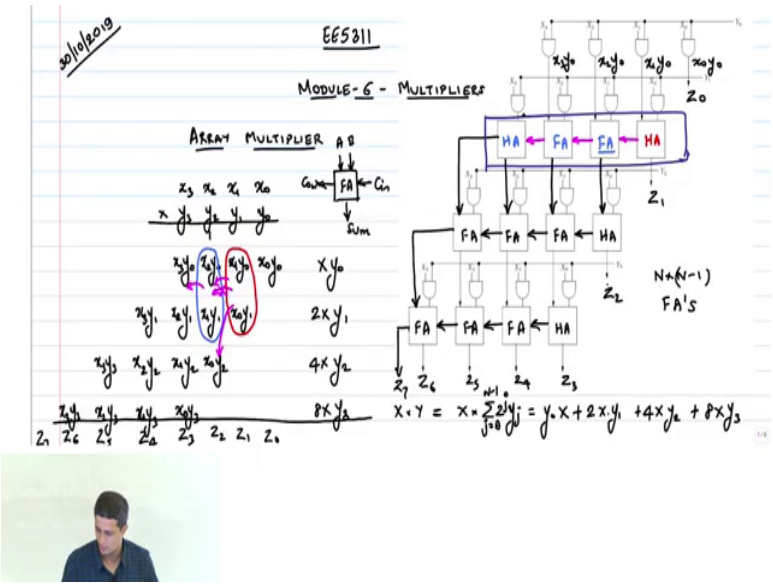


So, the learning objectives for this module right adders and multipliers designing full adder with least PMOS stack size using self duality or mirroring principle, constructing adder architectures to reduce delay from O of N to O of root N right O of root N up to that we did square root adder O of login you have to do carry look ahead adder in your project.

Then timing diagrams to show signal propagation of various adder, that is static timing analysis arrival times right and why we get benefits, then array multiplier for both signed and unsigned multiplication that is what we did today, then optimize the array multiplier using the inverting property that is what I told you, where if you use a NAND gate instead of AND gate then you can use this inverting property and optimize this further right. Then booth encoding unfortunately, this time I will not have time to cover it, but its effectively a way of reducing the number of partial products right.

Right now, if I have N bits N cross N multiplication, then I have N partial products to add right and each partial product right, if you look at each partial product here right.

(Refer Slide Time: 21:12)



If you look at the partial product in the multiplication here, they are just if any bit y 0 y 1 y 2 y 3 is 0, then the partial product also is 0 right. If it is 1 then the partial product is x right, it is x shifted by something right 2 power k into x which is shifted by something.

So, you are generating only two kinds of partial products 0 and x, but it is possible to generate other kind of partial products easily like for example, 2 x, this is just shifting right, shifting left right, you can exploit this in order to reduce the number of partial product additions right, where for example, if you treat this y 3 y 2 y 1 y naught as a base 4 number as opposed to a base 2 number.

Then you will get only 2 partial products, but the number of the kind of partial products that you have to generate will be different that is what booth encoding is all about just a very brief thing about what booth encoding is mathematically, we show why we can actually derive it and optimize it in a particular way there ok. You can look at my slides its very obvious ok, you take a look at my slides, it is all there, the math is worked out so, you can figure it out on your own as well ok.