Digital Circuits and Systems
Prof. S. Srinivasan
Electronics and Communication Engineering
Indian Institute of Technology, Madras
Lecture # 15
Array Multiplier

We have been looking at the circuits for arithmetic operations. We first considered the adders which are the most fundamental arithmetic operation and we have also talked about speeding up of adders, one example we took was the Carry Look Ahead then we talked about subtractors and subtraction being a complement of addition is taken into account while designing subtractors. Then the next most widely used operation in arithmetic computations is multiplication so we would like to see multipliers. Of course there are different types of multipliers depending on the speed and hardware; it is always the trade off. More hardware faster multiplication is possible but you need to spend more money. If you want to go for a smaller circuit with less hardware then it will take more time.

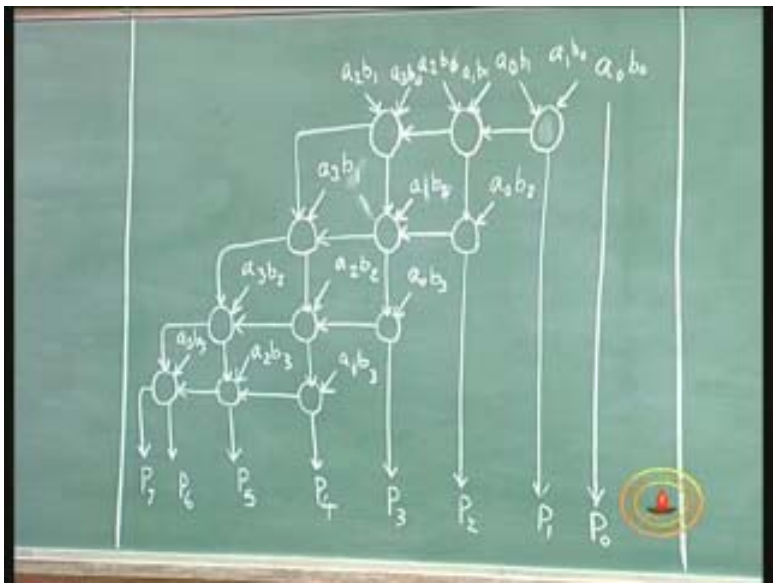
Basically the multiplication as you understand from our paper and pencil method is to do this shift and add that is we take the two operands one is called multiplicand the other is called multiplier and in case of decimal numbers each digit of the multiplier is multiplied with the multiplicand you write the result which we will called partial product then we take the second digit of the multiplier and multiplier again with the same multiplicand write it down from the first partial product. But when we do that we do the shifting because we have taken the second position of the multiplier, while we write the partial product we write it with a left shift of one digit position to the left and we continue to do that till we exhaust all the digits of the multiplier and finally we add the whole thing to get the final product. This is an algorithm which you are familiar within our hand computations, it is called shift and add algorithm because we have to add it after shifting it. All the partial products are added together to get the final product but before doing so we have to shift each of the partial product to one bit position to the left.

Of course the other method of doing it would be to add the multiplicand multiplier times. That is if you want to multiply a number n by another number m I can add n m times then also you can deploy but that is a very slow process and I need to do n additions. The shift and add is by far considered to be much faster and a more efficient method of multiplication which has been used in computers. But as we go for higher and higher speed whether the shifting has to be done one at a time or all at the same time because suppose we have only one adder we can get one partial product keep in a register or keep it in a storage then get the second partial product shift it and add it using one adder then that result I will keep in a storage then get the second partial product or third partial product again add it with the same adder so with one adder in theory I can accomplish the shift and add algorithm and if you have enough storage facility to store the multiplier, multiplicand and the partial results I can accomplish the whole multiplication in a series of additions one after the other.

On the other hand if I have a series of adders if all the partial products can be add at the same time I can get the result much faster because all adders can be added, all the partial products can be added the same time. So, one is the serial operation one after the other. The partial products are added one at a time till we complete the process of partial product addition. The other is to have all the partial products feeding on to the adders at the same time and then getting the sum in one stroke. Such multipliers are again as I said expensive in terms of hardware but faster. So it is always the conflict between the hardware you want to spend you want to use and the time you want to allow.

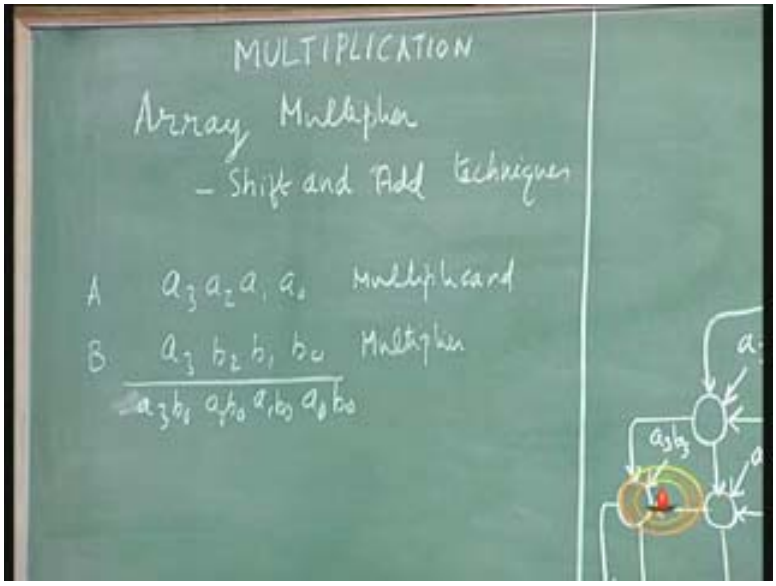
Of course this is only a simple scheme there are other schemes in which you can even make these things faster. For example shift and add can be made slightly faster by some innovation. Similarly this all parallel all adders at the same time can also be speeded up by some innovation. So speeding up like this Carry Look Ahead from the straight adder to the Carry Look Ahead adder and many other adder schemes which are faster we can also have multiplier schemes which are faster than the basic things. In basic thing though there are only three things we can recognize; one is the repeated addition which is inefficient, we have the shift and add with a set of adders, we repeatedly do a shift and add in which we use several adders where all work at the same time.

(Refer Slide Time 7: 47)



These are three schemes within each one we can speed up using many techniques which we are not going to get into this course. So what we will do today is to look at a circuit which will do the shift and add all at once such a multiplier is called an array multiplier or a parallel multiplier. Why is it called an array multiplier because it needs an array of adders. Array is lot of elements. So when we look at the circuit diagram we will know why it is called an array multiplier. Even array multiplier uses only shift and add algorithm. That means I have to have the numbers put in the proper position with appropriate shifting in order to get the final result which is correct.

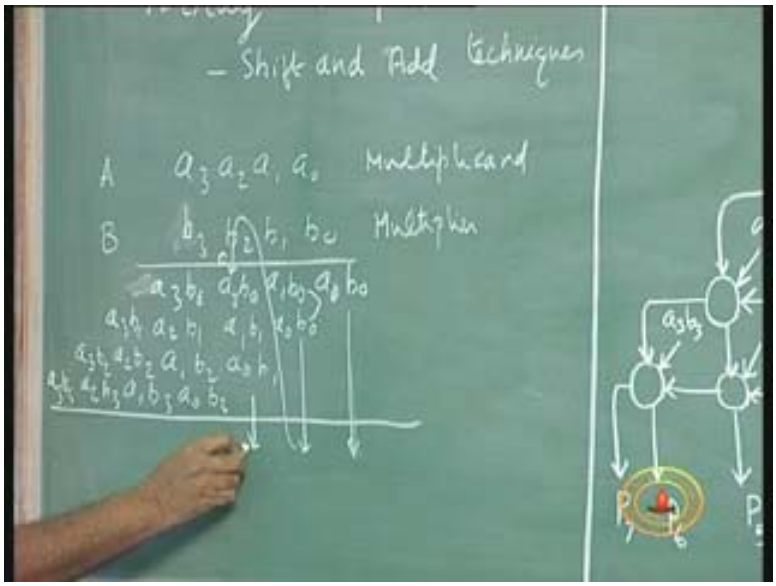
(Refer Slide Time 10:52)



Let us take a simple example of two 4-bit numbers. Let us say I want to multiply two numbers A and B each four bits a_3 a_2 a_1 a_0 is my first number and b_3 b_2 b_1 b_0 is my second number I multiply, all As and all Bs are either 1 or 0 these are binary numbers so As and Bs can only take values of 0 and 1. So first I have to multiply in a traditional multiplier paper and pencil multiplier this is not multiplication. Here (Refer Slide Time: 9:40) this is called multiplicand and this is called multiplier. In the decimal system you take one digit of the multiplier and multiply it with the multiplicand and write the result which is called partial product because this is not a complete product so I will call it partial product or intermediate product. Then I take the second position of the multiplier again multiplying this by the same number and write the partial result but with a shift. I will keep on doing it will have exhausted all the digits of the multiplier and I have this partial products with proper shifting then add column by column. This is we have do it. Now we will do exactly the same way here except these multiplications are only one bit multiplications.

So if I multiply this by b_0 this will be a_3b_0, a_2b_0, a_1b_0, and a_0b_0 that is the first partial product and the second partial product will be obtained by multiplying a by again b_1 which will be a_3b_1, a_2b_1. a_1b_1, a_0b_1 it is the same thing except that I have to shift it corresponding to the second bit position, from b_1 I need to shift the partial product when I write it down. Third is I will go to multiply this same a_3 a_2 a_1 a_0 by b_2 I will shift it again a_3b_2, a_2b_2, a_1b_2, a_0b_2 and finally this will be b_3 (Refer Slide Time: 12: 01) a_3b_3, a_2b_3, a_1b_3, a_0b_3 add a over-up so this will result in one number and this is same as a_0b_0. Then this would be added I will write the result here, this also can have a carry because this is a bit and this is a bit a_0b_0 can be 1 or 0 again.

(Refer Slide Time: 12:57)



Each product a_0b_0 and a_ib_j is a bit 0 or 1 so it can be used as 0 or 1 and if both are 1 there will be a carry. Thus I need to put the result and take the carry into this (Refer Slide Time: 12:51) add these three then take the carry here put the result here take the carry here put the result here take the carry here put the result here take the carry here put the result here, take the carry here put the result here, take the carry here put the result here and I may get one more carry finally.

Because in this one bit the carry of this will go into this and that can itself generate a carry. So how many bits will be there in the product? There will be 1 2 3 4 5 6 7 so there can be an eighth bit because there can be a carry here. So we have two 4-bit numbers and the result is it can be up to eight bits not necessarily eight bits but it can be depending on the numbers. Depending on the magnitude of the numbers the number can have fewer bits but the maximum number of bits you can have eight bits.

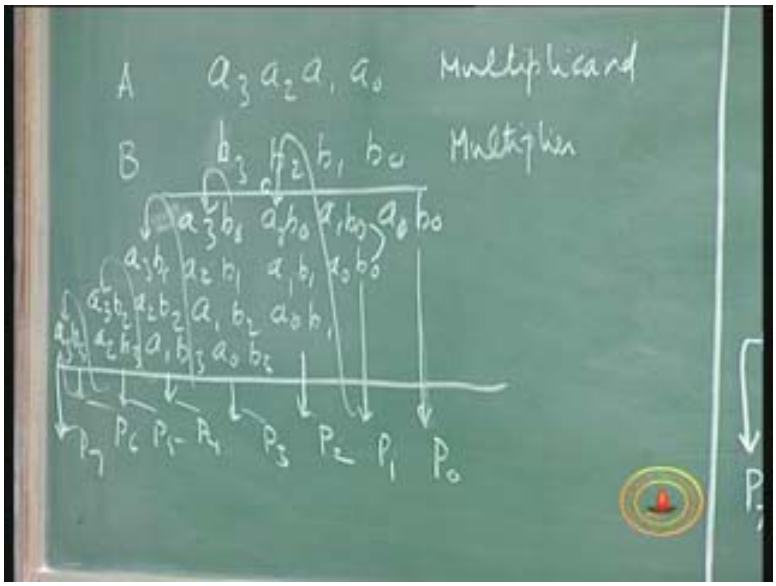
Thus I will call this P_0, product bit one P_1, product bit two P_2, P_3, P_4, P_5, P_6, P_7 this is P_2, this is P_3, P_4, P_5, P_6, P_7. So it is eight bits from P_0 to P_7 so a maximum of eight bits can be there. The first thing you should know about multiplication is when you multiply two numbers the total number of bits that are resulting is that the sum of the number of bits will be two operands. One operand is called multiplier and the other operand is called multiplicand bits. The sum total of the number of bits in these two numbers will be the maximum number of bits you can have in the product.

In the case of addition we saw that when we add two numbers of the same number of bits the result can exceed by one bit. Here it is the sum these two. Now the question is multiplication is very simple.

As I said I have to add a_3b_0 a_2b_0 a_1b_0 a_0b_0 with a_3b_1 a_2b_1 a_1b_1 a_0b_1 at the first step. If I

have a 4-bit adder with an appropriate shifting of the bit positions I can carry out this first partial product summation then I can use the same 4-bit adder and shift the results appropriately to get the sum of these two. Whatever I got as the sum of this can be added to this, I can repeat it. So I can do repeated additions using the same adder where I will take one row of the partial product another row of the partial product first complete the addition take the result feed it as the input take one more row of the partial product add them get the result put it as the input and take one more row of the partial product and complete the addition. In the algorithm both techniques are shift and add, shift and add with one adder not a single bit adder, since I need to add four bits I need to have an adder which is a 4-bit adder.

(Refer Slide Time: 16:32)



On the other hand I can have at the same time I cannot have these two numbers the result of this can flow into this the result can flow into this I can have many adders so that I can complete additions as fast as possible. This second version is called the array multiplier because I need an array of adders, rather than a single adder I need an array of adders that is called array multiplier, that is the most widely used multiplier today. When the cost of these devices as technology has improved, earlier we used to do this shift and add, in fact it was so expensive they used to take one single bit adder do the first bit addition and then the second bit addition then the third bit addition and complete one row and go to the next row that is called bit serial.

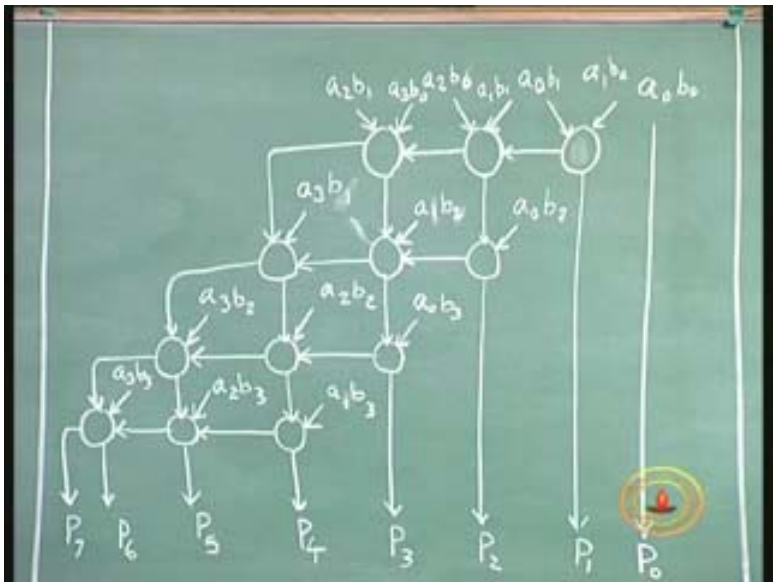
Bit serial addition is bit by bit serially I will add. Then we go to this second stage where I have a parallel adder with four bits or six bits or eight bits or whatever is the number of bits I need to complete my partial product addition of one row to another row. I will take row by row so row serial. Instead of bit serial I will have a row serial now a shift and add hardware. But now I don't even have to do that because the technology has become so inexpensive, the cost of these things are not at all concerned today, the size is not concerned, specification is not a concern, we are able to have large ICs Integrated

Circuits in which all of these things are packed. When in that case why do we have to do all this one by one and delay the whole process. It is a question of resources. If we have enough resources to complete my project I can get it done as fast as possible of course under certain constraints assuming that all of them can be done parallelly. I cannot commit a lot of resources and do a project really fast if my project implementation is a serial sequential, the first step has to be completed before the second step is taken up. If that is the requirement of my project I cannot do it even though I have lot of resources.

In this case we are assuming that all these additions can be carried out the same time, we have enough number of adders these are called product terms each individual term I will call it a product term a_0b_0 a_1b_0 a_2b_0 a_3b_0 each one of them is a product term so I will feed all these product terms with appropriate positions, with appropriate shifting in my adder array that's why it is called array multiplier, array of adders and we get the result as fast as possible, such a multiplier is called an array multiplier. But how are you going to get a_0b_0 as these are all AND gates right? When you have two numbers 0 and 1 multiply by another number which is also 0 and 1 the only case the result will be 1 is when both are 1 and in all other cases the result is 0. Is it not as same as the AND operation? I have a and b, a_i and b_j. My individual product term a_ib_j will give me a_ib_j if both are 0 or if one of them is 0 the output is 0 and if both are 1 the output is 1. So each of these product terms can be generated using an AND gate.

So first resource I need to commit for my multiplier is as many AND gates as possible. How many? I need to generate ai and bj for each I where i is equal to 0 to 3 and j is equal to 0 to 3 so four times four is sixteen product terms I need to generate so I need to have sixteen AND gates. Assuming I have sixteen AND gates in which I will appropriately feed a_0 a_1 a_2 a_3 and b_0 b_1 b_2 b_3 proper combinations I will get a_ib_j where i can vary from 0 to 3 and j can vary from 0 to 3. So in other words I can have from a_0b_0 to a_3b_3 all these sixteen terms are available to me at one stroke. With that assumption I will go and show you how to do the addition process. This is the circuit for which you will do this.

(Refer Slide Time: 21:04)



This is the circuit of the array multiplier. I have already assumed that all these product terms are available to you using AND gates. If you want to just show this then in one case you can show this. I didn't want to clutter this drawing with all the AND gates in each of this so I will just put an arrow with this product term. We have to assume that each of this product term is coming from an AND gate like this. So this AND gate will give you a_0b_0 and this a_0b_0 does not require any more additions, the last row the least significant column the least significant column this becomes a partial product b_0, the first bit of the product term.

The first bit of the product term P_0 is nothing but the a_0b_0 so that is why as I said is nothing but b_0 here. If we take the second column I need to add a_1b_0 to a_0b_0, a_0b_0 and a_1b_0 have to be added, a_0b_1 a_1b_0 and a_0b_1 have to be added these two product terms are available to me I put it in an adder and get the result this adder needs to be a half adder because I have only two terms and no carry, I have two inputs and no carry so I need a half adder and the sum goes to P_1 the carry is passed onto, the carry from this column has to go to the next column remember that.

When I add the partial product of this column to get the product term corresponding to this column the carry from here should go into the next column so my carry has to travel to the next bit position horizontally. This is the carry of this adder this is the half adder I put with the symbol H for half adder and this is sum (Refer Slide Time: 23:45). In the third column I will have to first add a_2b_0 to a_1b_1 along with the carry from the previous column and then to that sum I should add a_0b_1 take it as my P_2 term the product term, the P_2 bit of the total product and if there is any carry resulting from that I need to go to next stage. So there may be a carry resulting from here, there may be a carry resulting from here. Since there are two additions two carrys can be done, each of these carrys have to

go to the next column. The carry is always one weight higher bit position wise in hierarchy. So the carry from here would go here and the carry from this has to go to this. That is why I am having here a_1b_1 a_2b_0 adding to the carry of the previous this sum goes into this to complete the product term P_2 I need to add a_0b_2 further but the carry has go to the next bit position so this is my carry (Refer Slide Time: 25:08) this is my sum and with this sum I have to add a_0b_2 to get the sum which is my P_2 and this carry into this. So the carrys have to go into the next positions and the sum has to travel vertically, the carrys have to travel horizontally.

Now I don't have to keep on explaining all these terms. you take each column and take two terms at a time, if there has been a carry from the previous position put it on the top of it add it get a sum if that is the last addition take it as the particular bit of the product but if it is not the last addition go to the next term in the same column keep on adding sum, sum, sum in all those columns in which it has to be added one after the other and any carry generated has to be pushed into the next bit position whatever at this stage.

So a_0b_1 a_2b_0 added to this previous carry gets a sum and to this sum a_0b_2 I get the product bit P_2. This carry goes into this I am having a_2b_1 a_3b_0 so this carry is also taken into account and it provides another carry which has to go into the next bit position a_1b_2 which is this sum and this will give me this (Refer Slide Time: 26:52) along with this carry. Make sure that you never have more than three inputs to be added because you can have two inputs is called a half adder and three inputs it becomes a full adder. You cannot have more than three. So this is a half adder because I have the sum from here and this new term, this is a half adder. All these are half adders, this has be a full adder because I need a carry in, these two terms, carry out and sum. A full adder will have a carry in, two bits as inputs, carry out, and sum as the output. So this is a full adder so full adder, full adder and full adder (Refer Slide Time: 27:40).

So a carry from here is pushed into this next column, a_3b_1 is the first value in this column, if you draw it this way it is convenient I could have always pushed this full adder here because this carry is coming in a new term. But then if you don't keep that symmetry of the drawing it is very difficult to understand what is happening. So it is better to map in your drawing the hardware the circuit for the schematic as exactly the same way as the bits appear in your multiplication. So this multiplication can be taken as the reference for drawing this. Otherwise I could have push it here it is not going to harm me because the carry anyway be there, the new term will be there and then I can push it then it becomes sort of confusing after some time. It is asymmetric as something else. This is better to look at this way now in this symmetrical.

This goes here, this carry goes here so this is a full adder, it is pushed to the next stage so this addition process the carry will go into this a_3b_2 will be added to this carry and this carry and then this will come to new and then finally this will be the last addition of these two terms that will go into this so this will be the last adder (Refer Slide Time: 29:32) where a_3b_3 will be put along the carry from the previous position and this may generate a carry because there is a bit here, a bit here and a bit here because it is a full adder so there

may be a carry here so you have to put one extra carry. So this is the sum of the last full adder and carry of the last full adder.
Therefore in this the notation is H the half adder, F is full adder, S is sum, C is carry, and this is the multiplier called array multiplier. This is the most elementary form of array multiplier. You can make changes in this and speeding up is possible. Also, we are not talked about signed numbers. Should the numbers be all unsigned numbers or signed numbers? If it is signed numbers are they represented as twos complement? How it is going to be handled. Because when you multiply with two numbers the sum of the product depends on the signs of the individual numbers. If one of the numbers is negative the product is negative and if both the numbers are positive or if both the numbers are negative the product is positive. The sign of the product is positive if both numbers have same sign. If the numbers have different signs then the product will be negative.

All these things would be taken into account. I am not getting into all those details. so this can be improved further for speeding up, can be improved for handling for signed numbers, this is for unsigned numbers. I can do it for 8 bits, 16 bits whatever you want.
So in general now I have used 4 by 4 multiplier this is called 4 by 4 array multiplier, this is the name. What is the hardware required? 16 AND gates and how many adders totally? It is 4 times 3 so 12 adders out of which 4 are half adders and 8 are full adders. It is 4 half adders and 8 full adders so a total of 12 adders.

(Refer Slide Time 36:45)

If you want to generalize this if you want to call it m by n then m is the number of bits in the multiplicand and n is the number of bits in the multiplier. The first number is called multiplicand and the second number is called multiplier. In this case b is n, n is 4 so it is b as n is the number of bits, for ‘b’ it is 4 and for ‘a’ also it is 4 so here in this case m is 4 and n is 4 but what I mean by m is number of bits of ‘a’ and number of bits of ‘b’ is called n. It is only a notation, you can do it the other way also or you can use p, q or x, y. But I am using this just to keep the notation straight.

So if you want generalize that there are m times n AND gates, there is a half adder in each row, one half adder in each row, each row corresponds to the multiplier bit as each multiplier bit gives rise to a product row, each multiplier bit gives rise to a partial product row so the sum of the half adder is same as the number of bits in the multiplier. So n is the number of half adders, what is the total number of adders? It is n rows but in each row since we are going to have only one less adder because of taking this into account (Refer Slide Time: 34:28) it is one less adder in each row one less than the number of bits so there are n bits there are only n minus 1 adders in each row but there are m rows so the total number of adders is n (m minus 1) (Refer Slide Time: 35:24) so the remaining is full adder. That means total adder is mn minus n so full adder is mn minus 2n or n into m minus 2 numbers of full adders. Therefore the number of full adders is the total number of adders minus number of half adders.

There is a problem here (Refer Slide Time: 36:12) this is mn minus 2n plus n becomes mn minus n, just generalize this so you have to idea now. Therefore I can do an 8 by 8 multiplier, the m and n need not be same that is why I have given you as m and n otherwise I would have put n and n squared and all that. So I can have a different number of bits for the multiplier and for the multiplicand. So, if you know this idea the number of gates and number of adders half adders and full adders you have an estimate of the hardware.

The beauty is the whole thing is available as a single chip not even a 4 by 4 but even bigger than that. And as I said this extra circuitry for speeding up, as I said speeding up always comes at the cost of something else remember that. And also the signed numbers the 2s complement numbers these unsigned multiplications, signed multiplier which speed up, larger number of bits are things that are available. Today fortunately it is hardware ICs so you can just plug it and use it, just plug it in your circuit and use the circuit for multiplication purpose. Now we are talking about this as an improvement from bit serial forget about it bit serial is olden days.

That is you take only one bit full adder give a_0b_0 or whatever it is, for example you want to take one addition. Let us take this addition a_1b_0 a_0b_1 you put a_0b_0 a_1b_0 a_0b_1 into the full adder get the sum and the carry you should have some way of storing the carry till the new bits are put it the same full adder because I have only one full adder you remember. To this full adder I will have to get these two numbers I should have a way of keeping this carry till that time adding it at the right time and also keeping the partial products in storage places and all that. It was necessary when adders were expensive.

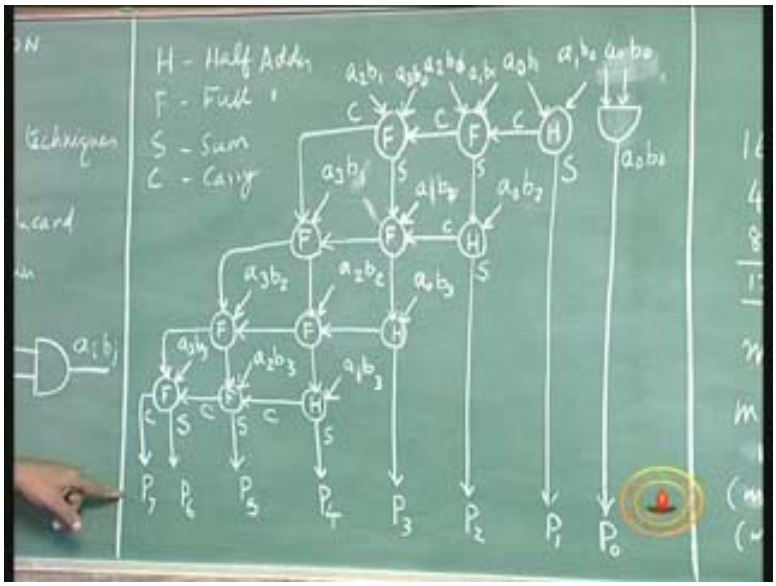
When adders became very cheap when the Integrated Circuit technology became so easy to fabricate and so inexpensive, now nobody talks of bit serial multiplier today but until recently the debate was between partial product and one multi-bit adder, not a single bit adder as in the case of bit serial adder but one multi-bit adder which will be repeatedly used for first adding this to this and then adding this to this and adding this to this (Refer Slide Time: 39:17) and adding this to this that is called as I said row serial.

Row serial was until recently being used because of the expensive nature of these array multipliers. Again as I said in the class room we can do a 4 by 4 and in the textbook you can find a 4 by 4 but you want to do a real job four bits is only sixteen 0 to 15 numbers and out of these you signed arithmetic it is plus or minus seven in the case of ordinary sign magnitude minus 8 to plus 7, it is such a small range and nobody used this as a multiplier, you can do it by mental sum, by tables.

Therefore we are talking of a 16-bit multipliers, 32-bit multipliers, 64-bit multipliers so it all exponentially grows the number of gates and full adders and all that. It becomes cost effective to do this because the speed has become such a critical issue today. We are talking of computers which have to work in nano second clock cycles. Therefore naturally I need to have hardware for multiplication which should be very very fast, and the only way to achieve is we have all of them available at the same time (Refer Slide Time: 40:52) so commit as much resources as possible to get the job done as quickly as possible.

Having said that is it all that fast we will have to see, because after all they are adders and adders suffer from carry propagation, we saw it in adders, carry propagation is a big issue. Here also there is carry propagation, this carry has to go here and then trigger this and this sum will go, of course there is no worry because until I have P_7 I cannot use this product. So let me see how quickly I can get P_7. I cannot do the full product until all the bits are in place.

(Refer Slide Time: 41:34)

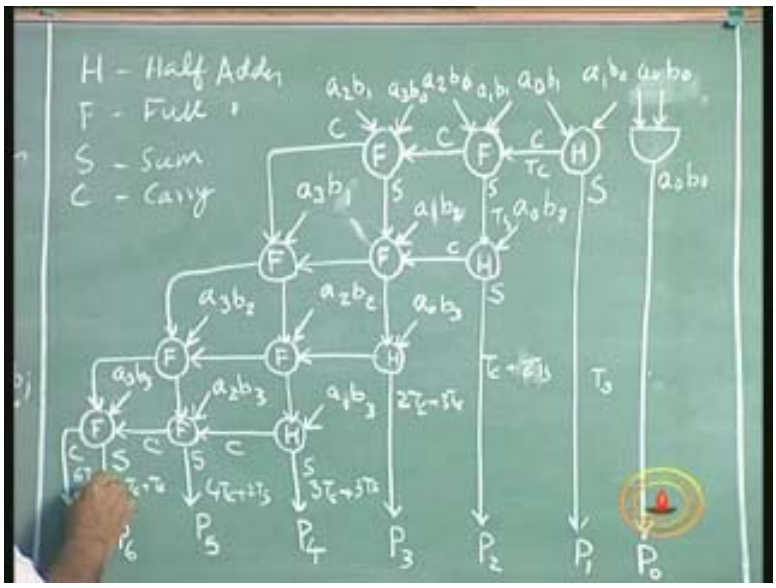


So, even though the sum may take more time once this carry is available here this would be fired and this sum will go here and carry will go here, I will not worry about this path anymore but I will worry about what happens to this carry how fast can I get it here. Similarly how fast can I get this carry here, how fast can I get this carry here how fast can I get this carry here, how fast can I get this carry here, how fast can I get this carry here

(Refer Slide Time: 41:59) and how fast can I get this carry here. So the number of carry is this 1 2 3 4 5 6. If I call T_c as the carry time or carry propagation time and T_s as sum propagation time of an adder. In fact for symmetry sake you have uniformity don't worry about few half adders. If we are talking about 64 by 64 or something like that you may talking of so many adders a small number of them needs to be a half adder and need not be taken into account, you can have them all as full adders with carry 0 that is not a big issue. So we will not worry about propagation time separately.

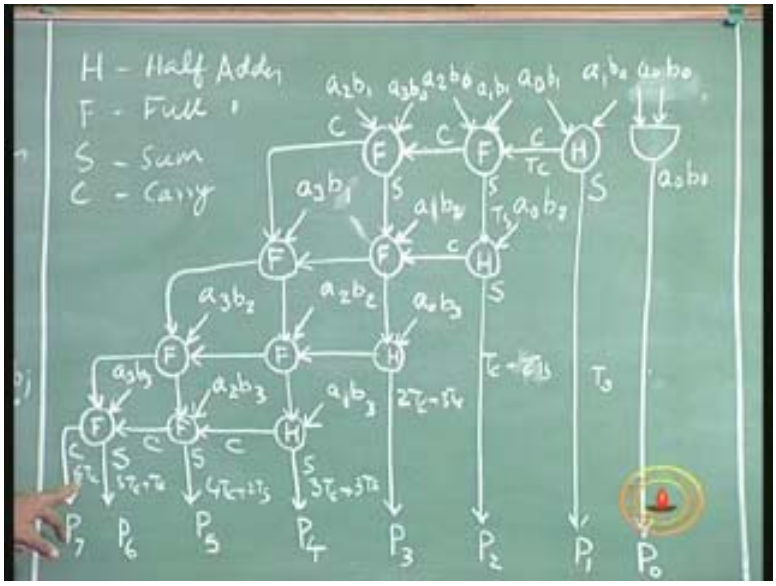
We will call it T_c the carry propagation time and T_s is the sum propagation time of an adder whether it is full adder or half adder we will assume this to be the same, the speed would then be here, this is available after T_c and this is available after T_s where T_s is the sum time. This is available instantly. I am not considering the add time. If I consider add time it is an extra time, T_A as the AND gate time. So T_A is common to all, all product terms are available after T_A so I will not worry about it now, finally I can add this T_A so I will have put T_A plus T_A and all that. So this is immediately available after T_A, this is T_A plus T_s and this will be T_C plus $2T_s$ (refer Slide Time: 44:05) this is $2T_c$ plus $3T_s$ because $1T_c$, second T_c, T_s, T_s, T_s and this will be $3T_c$ plus $3T_s$ and this is $4T_c$ plus $2T_s$ and this is $5T_c$ plus $1T_s$ and this is $6T_c$.

(Refer Slide Time: 44:50)



Therefore this term is available after six T_c after the multiplication has started, this term is available at $5T_c$ T_s after the multiplication has started etc. Usually T_c is more than T_s if you assume that. Of course you can always make an adder in which T_c is faster than T_s. So, if T_c is greater than T_s I will not worry about this T_{cs} because $6T_{cs}$ is the largest number of T_{cs} so the propagation time is $6T_c$ and the multiplication time is; T_A plus $6T_c$ and 6 is only a number for this particular combination in generated there would be n minus 1 plus m minus 1 one less for n and one less for m or this will be T_A plus this (Refer Slide Time: 46:30) this is the time at which I will get the final result.

(Refer Slide Time 46:41)



Or in general it is this (Refer Slide Time: 46:51). If on the other hand you have a circuit which is efficient because many times you have a circuit T_c is faster than T_s because T_c has to carry on and T_s is only local. T_s is a local phenomena, the sum propagation is a local phenomenon and carry propagation is the global phenomenon, it has to go. So because of that many times you want to optimize the circuits and it configures such that T_c becomes smaller than T_s, if that is the case what will be happen? If T_c is less than T_s then the propagation time will be T_A plus $3T_c$ plus $3T_s$ or in general it is T_A plus (n minus 1)T_s plus (m minus 1)T_c it is the same as this that is why I wrote both.

Here this is also important Ts. Even though this carry would have to propagate I have to wait for this. In $3T_c$ this is available after this, $3T_c$ is common between these two, this is $6T_c$ and this is $3T_c$ (Refer Slide Time: 48:39). But after this time for another $3T_c$ time this is available whereas for another $3T_s$ time only this is available. Now T_s is larger than T_c that means this P_4 is going to be the last not P_6 but earlier we said P_7 was the last to arrive. Now because $3T_c$ is common between these two $3T_c$ after this is available and P_7 is available slightly earlier because T_c is smaller than T_s whereas $3T_s$ is going to take a little more time so P_4 will be the last to arrive so I will have to take that into account again in n minus 1 m minus 1 form.

So why did I say all these about the speeds. I said I am committing a lot of hardware here. Instead of a row serial addition I am having an array of instantaneous addition. I hope to achieve instantaneous, moment the number A and B are given I want the product P which is A times B but of course it is limited by hardware which is AND gates and array of adders. We have to make sure that this is faster than this. of course there are techniques to speed up the process but as I said I am not going to discuss those things in this class but the point is you have to see whether the resources you are able to get whatever you want but if in spite of that you are not able to get this speed I don't want to use it. Before committing a lot of resource you have to make sure that you will get the job

done in the time required. If you cannot still meet the dead line what is the point in committing lot of resources to that. So I can think of another way of improving.

Can I get some more resources to speed it up or reach a point to know that technology cannot deliver what I want so I will go back and do the old version way, something like that?

With this we will conclude the arithmetic circuits. We talked about adders, half adder, full adder, multi-bit adders, fast adders, Carry Look Ahead adders. There are many other fast techniques which I have not considered in this class. We talked about subtractors and subtractor being 2s complement of adders. We talked about multipliers in different schemes and then one of them just to give you the flavor of the multiplication hardware.