

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


**Lecture - 54**  
**Ripple Adder Introduction**

So, as usual Module 6 let start with the learning objectives right; Adders and Multipliers. So, this forms the heart of your ALU. So, whatever we did till now is sort of to is a buildup in order to apply those concepts and optimize this particular module. The ALU that we build in a microprocessor has all its components coming from these concepts; adders and multipliers in general right. If you want to do convolution, then you have to do multiply accumulate right. You want to do any other operation which multiply accumulate to the very common thing.

So, typically the addition and that is the accumulation and the multiplication are operations that have been studied very extensively in the literature. Various architectures have been sort of you know invented in order to reduce the delay. Every picosecond that you gain from optimizing the accum, multiply accumulate operation or the accumulator in general will give you significant savings in the performance of the microprocessor, right.

So, therefore you will find various architectures that squeeze every single picosecond out of this operation and that is what we are going to study in this particular module ok, what are the Learning Objectives right.

(Refer Slide Time: 01:48)



### Learning Objectives

- ▶ Design a full adder with least PMOS stack size using self duality principle
- ▶ Construct adder architectures to reduce delay from  $O(N)$  to  $O(\sqrt{N}) - O(\log_2(N))$
- ▶ Draw timing diagrams to show the signal propagation of various adders
- ▶ Design an array multiplier for both signed and unsigned multiplication
- ▶ Optimize the array multiplier using the inverting property of a Full Adder
- ▶ Derive the Modified Booth Encoding to reduce the number of partial products
- ▶ Design and implement a multiplier based on the Modified Booth Encoding algorithm

Janakiraman, IITM EES311- Digital IC Design, Module 6 - Adders and Multipliers 2/58

So, first is you have to be able to design a full adder with the least PMOS stack size using self-duality principle. This we already saw glimpse of in the earlier thing where I showed you the mirroring circuit, right.

So, this is something we have to be able to do. Construct adder architectures to reduce the delay from  $O$  of  $N$  to  $O$  of root  $N$  to  $O$  of  $\log N$ , right.  $O$  of  $\log N$  is going to be achievable only through something known as a Carry Look Ahead Adder which I am not going to cover here. It is a self-study part and you will have to implement it in the project. So, please go and look at it, study it on your own and implement a circuit and show it in your project right. Then you should be able to draw timing diagrams to show signal propagation of various adders, right.

What a timing diagram in the sense? You have you have to show the arrival times at various points in the circuit and how and the optimization of the architecture is going to happen in order to sort of save or not waste the arrival times that you see there, ok. Then you should be able to design an array multiplier for both signed and unsigned multiplication. So, we will also look at unsigned multiplication where the numbers will be represented in 2's complement form, right. So, we look at the math behind the 2's complement multiplication, then we will see how an unsigned multiplier can be converted to a signed multiplier with very minimal modifications and the math behind it.

So, that it is you know it is consistent right then you should be able to optimize the array multiplier using the inverting property of a full adder right. I will deal with this today. Booth Encoding again I really do not know I will have time, ok. If I am running out of time I may not cover it this time, but it is not too complicated for you to go and figure it on your own as well ok.

Then Design and Implement a Multiplier based on Booth Encoding Algorithm. So, last two things depending on time I will see if I can cover it or not ok. So, great.

(Refer Slide Time: 03:51)

20/10/2019

EESB11

MODULE-6. ADDERS

Full Adder:

	A	B	C <sub>in</sub>	S	C <sub>out</sub>	C <sub>out</sub>
(DEL) KILL →	0	0	0	0	0	0
(DEL) KILL →	0	0	1	1	0	0
PROP →	0	1	0	1	0	C <sub>in</sub>
PROP →	0	1	1	0	1	C <sub>in</sub>
PROP →	1	0	0	1	0	C <sub>in</sub>
PROP →	1	0	1	0	1	C <sub>in</sub>
GEN →	1	1	0	0	1	1
	1	1	1	1	1	1

Diagram of a Full Adder block:

```

graph LR
    A --> FA[FA]
    B --> FA
    Cin --> FA
    FA --> S
    FA --> Cout
  
```

Logic Equations:

$$D = \bar{A}\bar{B}$$

$$P = A \oplus B$$

$$G = AB$$

Not A Full Adder

So, let us start with the concept of an adder and I am going to look at a full adder directly.

So, a full adder is a block that takes three inputs A B full adder right and it takes a carry in and gives out a carry out and of course, it also gives you a sum ok. So, let us let us put down the truth table of these signals sum carryout 0 0 1 0 1 0 ok. So, what is the sum? It is essentially just A XOR B XOR C, right.

So, this is basically 0 1 1 0. So, if there are even number of 1s you get 0 right and if it is an odd number of 1s you get a 1 1 0 0 1. What about Carry out ? 0. Essentially the carry out is you are adding the two numbers. If you have 1 and 1 right if you have more than 2 1s or if you have 2 1 2 or more 1s, then your carry out you have to take

it to the next bit right. Like just like you do in decimal numbers. Only if the number goes beyond 9, you do it there. Here you do it if the number goes beyond 1.

So, therefore this will be 0 0 0, there are two 1s here, you get 1 again 0 1 1 and 1 ok. There are certain other intermediate signals that are also very useful and we will look at them, ok. So, let us look at what happens to the carry in, in certain cases. So, let us say that my carry in is 0 or 1, but my inputs are 0 0. What happens to the carry out in that case?

Student: 0 0.

0 0, right. It means that irrespective of the input carry the output is it just killing the carry, right. The carry is just gone because the inputs are basically 0 0. So, this is what we are going to call kill ok K.

Now, what happens if the input any one of them A or B happens to be 1, right? That is basically if you see these cases, this is let me put this here. Sorry in black it is kill right or delete, delete ok. So, I will call this D. So, what happens in the case of the blue arrows that any one bit A or B happens to be 1, what happens to the carry out? Carry out equal to?

Student: Carry in.

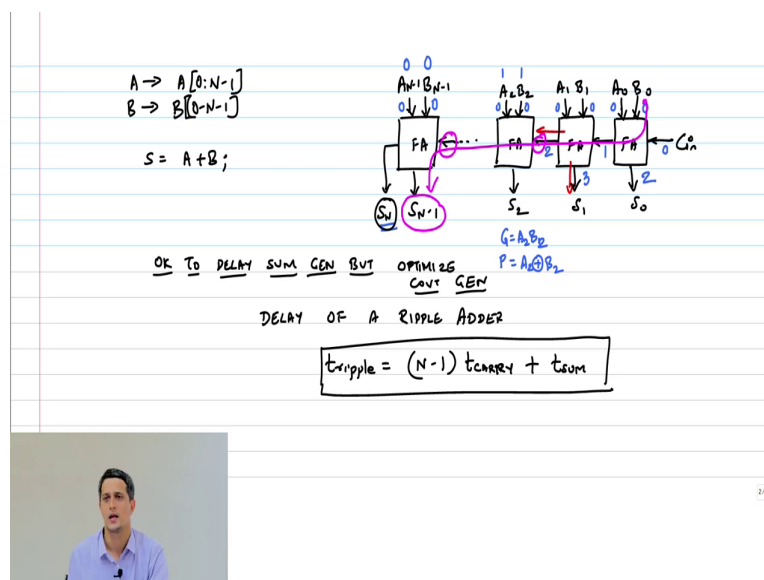
Carry in, right. This is nothing but let me write this here again slightly differently. This is 0 0, this is C in right, therefore this is nothing but propagate, the carry is simply being propagated from input to output. What about the last two cases? If both A and B happened to be 1 irrespective of what the carry in is carry out is being generated, it means those two bits are simply going to generate a carry irrespective to what the input carry is.

So, therefore this is called generate right. This is basically 1 1. So, therefore I can now define some intermediate signals, delete propagate and generate. What is the logic function for delete?

Student: A bar.

A bar B bar. What is Propagate? A XOR B, any one bit happens to be 1, it goes, A XOR B. What is generate? AB. Key thing to note is none of these signals are functions of C in ok. So, we will see why we are even looking at that case. Why is A and B being treated differently from carry in. For that we have to look at what we want to do in practice. In practice we want to take the full adder and use it, so that we can compute the addition of N bits at a time.

(Refer Slide Time: 09:54)



So, A and B are now going to be N bit vectors. A is some A of 0 to N minus 1 and B is B of 0 to N minus 1. I want to now get the sum equal to A plus B. This is not the logical OR this is the summation right. I want to create the sum of this and get the carryout also because it is an over flow, then I need to set that bit as well.

So, what do you do? You take this full adder that we created and cascade them one after the other right. So, this is my  $A_0 B_0 A_1 B_1 A_2 B_2 \dots A_{N-1} B_{N-1}$ , right and this is going to be my carry in and for you can assume that it is separate bit or it is grounded does not matter.

The resultant out of these full adders will be  $S_0 S_1 S_2 \dots S_{N-1}$ , all the way to  $S_{N-1}$  and the carryout will form the bit  $S_N$ , right. See add 2  $N$  bit numbers, you can get an  $N+1$  bit answer, right. So, this will form  $S_N$  plus 1, sorry  $S_N$ . I am sorry  $S_N$   $N+1$ th bit, right. This is the carryout. All these are full adders, all I am doing this taking the carryout right full adder and cascading it like this.

So, now you look at what happens to the arrival times. This is something we did last class right. This static timing analysis of propagating signals. Let us assume for now that the carry in right takes carryout generation takes 1 unit of time, carry in generation time takes I mean the sum generation takes 2 units of time, let us assume, ok.

So, what happens first of all, what is an arrival time of all the  $N$  bits?  $A$  and  $B$ . They are all 0. So, the key thing is all the  $N$  bits arrived at the same time they are all available at time 0 right. Carry in also, let us assume it is available at 0 ok. So, what is at what point will  $S$  naught be ready? 2. What about the carry out of the first block? 1 unit. When will the carry out of the second block be ready?

Student: 3 units.

3 right and what about the sum?

Student: (Refer Time: 13:11).

1 plus?

Student: 3.

So if this is 1, sorry this will be 2 right?

Student: 2

Yeah 2, correct. Now what about so you go you keep doing this, you will find that my carry is being propagated from one stage to another stage which means that the  $k$ -th stage cannot finish its evaluation until the carry in has come right which has to come all the way from somewhere.

So, this carry has to propagate from the 0th bit to the  $n$ th bit in order for the final result to be ready. So, in some sense while A and B are ready straightaway they are available at time 0 across all the  $N$  bits, right. So, that is why what we are doing is this carry bit is more timing critical than the bits A and B in general, ok.

So, now let us see what happens. Suppose  $A_2 B_2$  happen to be 1, right.  $A_2$  and  $B_2$  happened to be 1 1, thus the carryout let say which is  $S_N$ , right, this one  $S_N$  does it depend on C in now? No. Why because that stage has generated a carry. So, the generate signal which is here in stage 2 which is  $A_2 B_2$ , you can just evaluate this even before that actual carry comes in right.

So, this is a reason we are trying to create this intermediate signals just to make a decision on whether the carry you need to wait for the carry or not ok. Here we are not making a decision. We are just evaluating it. Later you will see that I can make a decision based on this information, ok.

Similarly, the propagate here is  $A_2 \text{ XOR } B_2$ , right. If that happens to be 1, that means the carry has to be propagated which means that you have to wait for that carry to come. On the



other hand if G happens to be 1, then I can say that this from this stage onwards everything can just be evaluated with that generate signal.

I do not have to wait for the carry in. This is the idea the intuition behind creating this intermediate signals, generate, propagate and delete. Of course if that if A and B and happen to be 0 0, what happens to the SN? Obviously it is 0. It has simply kill the carry. It does have to wait for C in to propagate all the way, right.

So, first thing is what is the critical path delay of this circuit? What is the critical path delay of this circuit?

Student: (Refer Time: 16:29).

So, is it SN or is it SN minus 1?

Student: (Refer Time: 16:39).

So, you see what is happening right where is arrival time 0 0 S naught is available at time 2, carryout is available at time 1. So, that sum takes one more unit ok, but the idea is while S1 is being generated, you propagate the second carryout into the next stage. That is why it is important to generate the carry before the sum.

So, we said that assume that carryout generation time is 1 unit and the sum generation is 2 units. Why did I make this discrepancy there? It is because the sum can take extra time to generate, but the carry has to propagate. So, while this carry is propagating from this stage, this sum is being evaluated in parallel. You understand? So, that carry is going to the next stage that evaluation is happening while that is happening this sum is already evaluated.

So, it is ok to delay the sum a little bit, but propagate the carry in as soon as possible because that is what the sum is not going to hold up the next stage is evaluation, the carry will right. So, I am saying it is sort of dead lock argument unless I tell you why I gave you this 1 and 2,

you cannot argue with this critical path correctly. So, I am just trying to resolve that dead lock. The carry is more important than the sum because that is propagating sum is not propagating and therefore, I have to it is ok to delay the sum in general ok.

So, it is ok to delay sum generation ok. So, the maximum arrival time is obviously going to be for this particular bit. Now in order now what is it going to depend on, which will which arrival time is that going to be dependent on? You have  $A_{N-1}$   $B_{N-1}$  and the carry in.

Student: A plus B.

Obviously this is going to be the max arrival time then this will be the max arrival time. So, the critical path will basically be like this right. So, if you look at the delay of ripple adder right,  $t$  it is going to wait till  $N-1$  carries are propagated right and the last sum is evaluated.

What I am saying is  $SN$  will get generated before  $SN-1$  because of the way we have designed the adder, carry comes before the sum, right. So, this is  $N-1$  times  $T_{\text{carry}}$  plus  $t_{\text{sum}}$ . It is not  $N$  times  $t_{\text{carry}}$ ; is all I am saying ok. So, here you can now substitute for  $N$  equal to whatever value and you know carry equal to 1 sum equal to 2, you will find the arrival time and you can check for yourself. You construct 8 bits and just see if the arrival time is correct or not, ok.

So, now this tells us that when we are creating our circuit for the full adder, we have to prioritize the carry out circuit over the sum circuit because ripple adder delay goes as  $N-1$  times  $t_{\text{carry}}$  plus only 1  $t_{\text{sum}}$  right. It is to delay some generation, but optimize  $C_{\text{out}}$  generation, that is the key conclusion that we come to after this quick analysis, right.