

**Computer Architecture and Organization**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

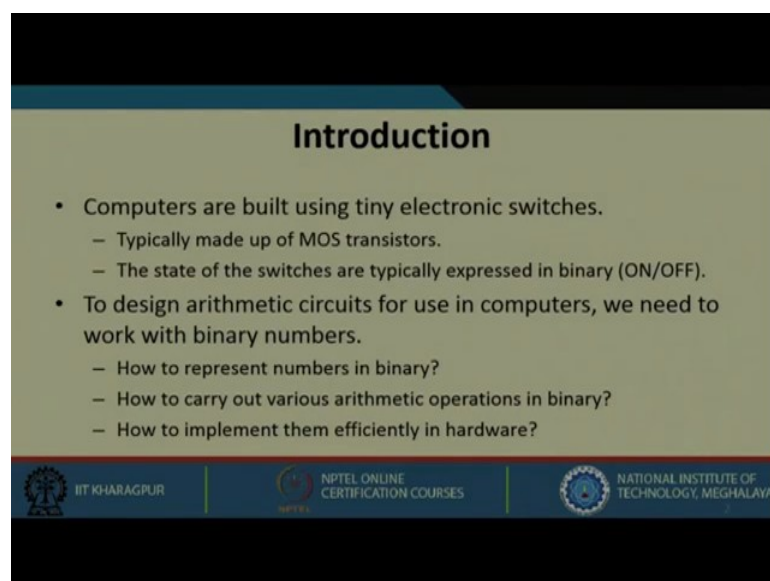
**Lecture - 33**  
**Design of Adders (Part I)**

So in this course we have so far seen various aspects of computer organization and architecture. So, if we recall we looked at the instructions and architecture of a typical processors. As a case study we looked at the MIPS 32 processor, then you looked at the design of the data path and how we can design the control unit of a typical processor or machine, then we looked at various aspects of memory system design.

Today starting from this lecture we shall be discussing various aspects of designing the arithmetic logic unit of a computer system. So, as you know a computer system essentially is meant to do some computation and in that sense the ALU or the arithmetic logic unit forms the basic heart of the system.




We shall be starting by looking at how we can design the different kinds of circuits for implementing addition, multiplication, division etc. The topic of our lecture today is design of adders.

(Refer Slide Time: 01:40)



**Introduction**

- Computers are built using tiny electronic switches.
  - Typically made up of MOS transistors.
  - The state of the switches are typically expressed in binary (ON/OFF).
- To design arithmetic circuits for use in computers, we need to work with binary numbers.
  - How to represent numbers in binary?
  - How to carry out various arithmetic operations in binary?
  - How to implement them efficiently in hardware?

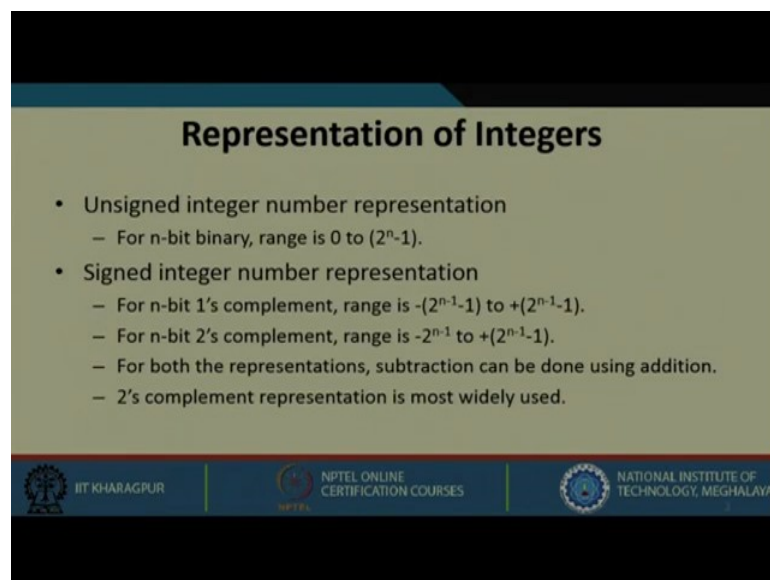
 IIT KHARAGPUR       NPTEL ONLINE CERTIFICATION COURSES       NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

As you know that computers are built using tiny electronic switches the switches are typically in the form of MOS transistors now the way this MOS transistors work in a digital circuit or digital system is that they can be turned on or off, they can be in 1 of 2 states. So, quite naturally such a system can be used to implement or model a binary number system where the state of the switch can be mapped to a binary digit 0 or 1.

So, the essential idea here is that when you are talking about designing arithmetic circuits we need to work with binary numbers although in practice we are more familiar with the decimal number system the way we work we calculate on paper, but here with respect to computers we need to work with binary numbers.

To recall you can represent binary numbers in either an unsigned form or in signed form; with respect to signed number representation we have seen the sign magnitude 1's complement and the 2's complement representations. What we will see now is that how we can carry out various arithmetic operations in binary and how to implement these operations efficiently in hardware; this will be the main objective of the next few lectures.

(Refer Slide Time: 03:43)



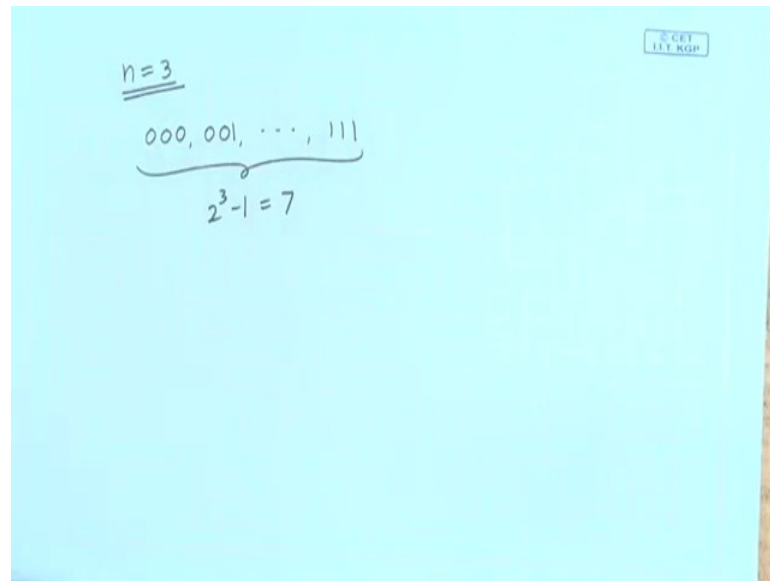
**Representation of Integers**

- Unsigned integer number representation
  - For n-bit binary, range is 0 to  $(2^n-1)$ .
- Signed integer number representation
  - For n-bit 1's complement, range is  $-(2^{n-1}-1)$  to  $+(2^{n-1}-1)$ .
  - For n-bit 2's complement, range is  $-2^{n-1}$  to  $+(2^{n-1}-1)$ .
  - For both the representations, subtraction can be done using addition.
  - 2's complement representation is most widely used.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, we make a quick recapitulation of the representation of integers. You may recall that with respect to unsigned number representation. Let us say we are representing integers. So, in n-bits we can represent numbers from 0 up to  $2^n - 1$ .

(Refer Slide Time: 04:10)

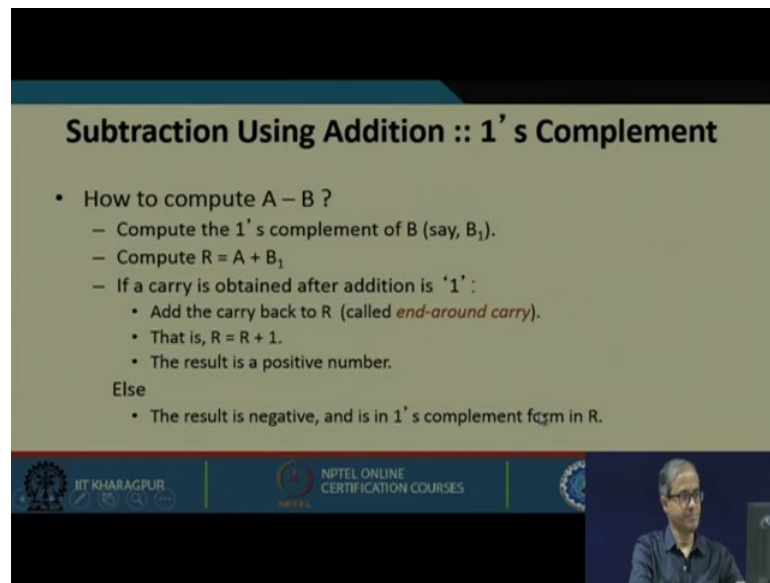


So, let us say suppose  $n = 3$ . So, for  $n = 3$  you can represent numbers starting from  $0\ 0\ 0$  up to  $1\ 1\ 1$ . So, there are a total of  $2^3 - 1$  or 7 possible combinations.

This will be the range of the numbers, but when you talk about signed representation as it said there are several alternative methods out of them 1's complement and 2's complement are the most common. For 1's complement representation in  $n$ -bits the range of the numbers that can be represented is  $(-2^{n-1} - 1)$  to  $+(2^n - 1)$ , but for 2's complement representation when on the negative side we can represent 1 extra digit number extra number  $(-2^{n-1} - 1)$ .

The reason is that for 1's complement representation there are 2 alternate representations of 0 this we have already seen earlier, but in 2's complement form 0 has a unique representation. The main advantage of this 1's and 2's complement forms is that we really do not need a subtract circuit in our ALU in both these representation subtraction can be done using addition alone, but we will see for reasons that we will be discussing that out of these two methods, again 2's complement representation has a distinct advantage and therefore, it is most widely used.

(Refer Slide Time: 06:03)



**Subtraction Using Addition :: 1's Complement**

- How to compute  $A - B$  ?
  - Compute the 1's complement of B (say,  $B_1$ ).
  - Compute  $R = A + B_1$
  - If a carry is obtained after addition is '1':
    - Add the carry back to R (called *end-around carry*).
    - That is,  $R = R + 1$ .
    - The result is a positive number.
  - Else
    - The result is negative, and is in 1's complement form in R.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, let us first look at how we can carry out subtraction using 1's complement representation. The idea is fairly simple well if I subtract a number B from A, let us say  $A - B$  this is what we are trying to compute what we do we first compute the 1's complement of B let us call it  $B_1$  the 1's complement of B, then we add this 1's complement of B to A; that means, what we calculating A plus  $B_1$ .

Now after this addition if we find that there is a final carry out if the carry of 1 is coming out then what we do we make a correction, we take this carry back and add this 1 to R. That means, we are effectively doing  $R$  equal to  $R$  plus 1 as a corrective step whenever there is a carry of 1 coming out and in this situation the final result will be a positive number, but; however, if there is no carry coming out this will imply that the result is negative and it is already in 1's complement form in this variable or register R. Let us see an example let us say we are trying to subtract 2 from 6 in 4 bit representation.

(Refer Slide Time: 07:31)

**Example 1 :: 6 – 2**

1's complement of 2 = 1101

$$\begin{array}{r} 6 \quad :: \quad 0110 \\ -2 \quad :: \quad 1101 \\ \hline 10011 \\ \text{End-around carry} \rightarrow 1 \\ \hline 0100 = +4 \end{array}$$

Assume 4-bit representations.  
Since there is a carry, it is added back to the result.  
The result is positive.

IIT KHARAGPUR Spring Semester      NPTEL ONLINE CERTIFICATION COURSES Programming and Data      NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, 6 in 4 bit is 0110 and 2 is 0010. So, the 1's complement will be 1101; so we are actually adding the 1's complement of 2; that means, minus 2 to 6 directly. So, 0 and 1 is 1 no carry 1 and 0 is 1, 1 and 1 is 0 with a carry of 1, 1 and 1 is 0 with a carry of 1. So, there is a final carry out. So, what we are saying is that this final carry out we are bringing it back and adding to this intermediate sum. So, 0 0 1 1 plus 1 this becomes 0 1 0 0 this is the final result plus 4. This is the how subtraction is carried out when the result is positive.

(Refer Slide Time: 08:34)

**Example 2 :: 3 – 5**

1's complement of 5 = 1010

$$\begin{array}{r} 3 \quad :: \quad 0011 \\ -5 \quad :: \quad 1010 \\ \hline 1101 = -2 \end{array}$$

Assume 4-bit representations.  
Since there is no carry, the result is negative.  
1101 is the 1's complement of 0010, that is, it represents -2.

IIT KHARAGPUR Spring Semester      NPTEL ONLINE CERTIFICATION COURSES Programming and Data      NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us take another example where the result is supposed to be negative 3 minus 5. So, 3 is 0 0 1 1 and the 1's complement of 5 is 1 0 1 0 you just add them up 0 1 is 1, 1 1 is 0 with a carry of 1, 0 0 1 is 1, and 1 0 is 1 but no carry. So, when there is no carry, our first conclusion is that the result is negative and whatever is remaining here this will be the result in 1's complement form, you recall 1 1 0 1 in 1's complement representation of the number -2.

(Refer Slide Time: 09:25)

**Subtraction Using Addition :: 2's Complement**

- How to compute  $A - B$  ?
  - Compute the 2's complement of B (say,  $B_2$ ).
  - Compute  $R = A + B_2$
  - If a carry is obtained after addition is '1' :
    - Ignore the carry.
    - The result is a positive number.
  - Else
    - The result is negative, and is in 2's complement form in R.

The slide footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA.

So, this is what has been summarized here and what we get here. Now let us look at the 2's complement representation. So, in 2's complement also the idea is fairly similar. So, when you are trying to subtract the number B from A, we take the 2's complement of B.

So, you recall the 2's complement of order of a number is the 1's complement plus 1 you add an additional 1 to the number to get the 2's complement form. So, in this case this number which you are trying to subtract you take the 2's complement of that number. So, let us call it  $B_2$  and we add  $B_2$  to A this is how we carry out subtraction, and after this addition step if you see that a carry is coming out simply ignore the carry and your conclusion is that the result is a positive number, but if there is a no carry your conclusion will be the result is negative and it is already in 2's complement form.

So, you see here in 2's complement form that additional corrective step is not required that end around carry you are bringing it back and adding to the partial sum that step is not required here.

In a sense 2's complement representation is more efficient in terms of the effort of calculation. So, when you are talking about implementing in hardware this will also be directly responsible for the decision that will be taking that 2's complement will get better than 1's complement. So, let us take some examples again here.

(Refer Slide Time: 11:11)

**Example 1 :: 6 - 2**

2's complement of 2 =  $1101 + 1 = 1110$

6 :: 0110  
 -2 :: 1110  
 -----  
 10100 = +4

ignore carry

Assume 4-bit representations.  
 Presence of carry indicates that the result is positive.  
 No need to add the end-around carry like in 1's complement.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us take this 6 - 2. So, for 2 the 1's complement is 1 1 0 1 you add 1 to it you get the 2's complement it is 1 1 1 0. You simply add 6 and -2 0 0 is 0 1 1 is 0 with a carry of 1, 1 1 1 is 1 with a carry of 1, 1 0 1 is 0 with a carry of 1. So, there is a carry coming out you simply ignore this carry and 0 1 0 0 whatever remains that is your final result plus 4 as I said here you do not need to add the end around carry.

(Refer Slide Time: 12:00)

**Example 2 :: 3 - 5**

2's complement of 5 =  $1010 + 1 = 1011$

$$\begin{array}{r} 3 \quad :: \quad 0011 \\ -5 \quad :: \quad 1011 \\ \hline 1110 = -2 \end{array}$$

Assume 4-bit representations.  
Since there is no carry, the result is negative.  
1110 is the 2's complement of 0010, that is, it represents -2.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

When the result is supposed to be negative like the same example 3 - 5 you take the 2's complement of 5 this is the 1's complement plus 1, this is the 2's complement you add them up 1 1 is 0 with a carry of 1, 1 1 1 is 1 with a carry of 1, 1 0 0 is 1, 0 1 is 1 no carry. So, your conclusion is that your result is negative and 1 1 1 0 is the 2's complement representation of the result. Now you can check 1 1 1 0 is nothing, but minus 2. So, you are getting the correct result.

So, the idea is that for 2's complement representation whether you are adding a larger number from a smaller number or a smaller number from a larger number your addition mechanism is identical. Subtraction or an addition are no different when you want to subtract you represent a negative number in 2's complement form and you add it. So, you only carry out addition no subtraction is required right.



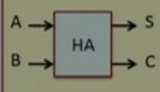
(Refer Slide Time: 13:07)

### Addition of Two Binary Digits (Bits)

- When two bits A and B are added, a sum (S) and carry (C) are generated as per the following truth table:

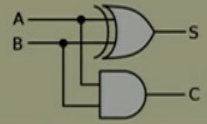
Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$\begin{array}{l} 0 + 0 = 00 \\ 0 + 1 = 01 \\ 1 + 0 = 01 \\ 1 + 1 = 10 \end{array}$$



$$S = A'B + A.B' = A \oplus B$$
$$C = A.B$$

**HALF ADDER**



The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter in the bottom right corner.

So, when we talk about addition of 2 binary digits let us say. So, when you are adding 2 bits A and B we generate a sum S and a carry C as per the truth table that is shown here. If the input bits are 0 0 this sum is 0 as well as carry is 0, if they are 0 1 or 1 0 then there will be a sum of 1, but no carry, but if they are both 1 and 1 then sum will be 0 and carry will be 1. So, this is the rule as I said sum plus these two numbers A and B this will be generating a sum and a carry. So, this is actually shown here 0 0 0 1 in that order first carry on then sum 0 1 and 1 0. So, this kind of a circuit which implements this truth table is called a half adder, and in a block diagram form we can show it like this A and B are the 2 inputs and this sum and carry are the 2 outputs.

So, if you just write down the expression from this truth table expression for sum will be  $AB' + A'B$  this is nothing, but the exclusive or of A and B ( $A \oplus B$ ). and C will be 1 when AB both are 1 that is AB. So, in terms of implementation a half adder will consist of an exclusive OR gate and a AND gate.

(Refer Slide Time: 14:50)

### Addition of Multi-bit Binary Numbers

0010110 ← Carry	1111110 ← Carry
0101011 ← Number A	0111111 ← Number A
+ 0001001 ← Number B	+ 0000001 ← Number B
-----	-----
0110100 ← Sum S	1000000 ← Sum S

- At every bit position (stage), we require to add 3 bits:
  - 1 bit for number A
  - 1 bit for number B
  - 1 carry bit coming from the previous stage

**WE NEED A FULL ADDER**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, when we talk about addition of multi bit numbers normally we do parallel addition like this is a number this is another number. So, when you add them we start from the least significant bit. So, initially the carry input is 0. So, this row shows the carry. This 1 1 will result in a sum of 0 with a carry of 1 this one 1 1 and 0 added again sum is 0 and carry of 1. 1 0 0 will give a sum of 1 no carry. 0 1 1 will generate sum of 0 carry of 1, 1 0 0 will give 1 no carry this again 1 no carry 0 0 0 0.

So, you get the sum you take another example here where this is an extreme case of something that I am trying to demonstrate. So, the first number is like this 0 1 1 1 1 and the second number is only a 1 less zeros. So, you add the first two binary digits get a 0 with a carry of 1, 1 and 1 will be 0 with a carry of 1, the same thing is repeating 0 with a carry of 1 0 with a carry of 1 and finally, this carry will be generating the last bit of sum.

So, what I tend to illustrate in this example is that the carry is propagating from the least significant stage to the most significant stage. After adding the first digit there is a carry generated, this carry again generates the carry, this carry again generates a carry. So, there is a rippling effect of the carry from one stage to the other. So, it is something like that from the least significant stage to the most significant stage the carry will be moving stage by stage in that fashion.

So, you will have to wait until all the carry has rippled through the different stages of addition and finally, after that you get the result. So, the observation here is that for

adding multi bit number for every stage or bit position we need to add three digits, two digits for the numbers A and B and one digit from the carry which is being generated from the previous stage. So, essentially we require to add three bits, one bit for A, one bit for B and one carry bit this kind of an adder which adds three bits is called a full adder.

(Refer Slide Time: 17:34)

**Full Adder**

Inputs			Outputs	
A	B	C <sub>in</sub>	S	C <sub>out</sub>
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Block Diagram: A box labeled 'FA' has three inputs: A, B, and C<sub>in</sub>. It has two outputs: S and C<sub>out</sub>.

$$S = A'B'C_{in} + A'B'C_{in}' + A.B'C_{in}' + A.B.C$$

$$= A \oplus B \oplus C_{in}$$

$$C_{out} = B.C_{in} + A.C_{in} + A.B + A.B.C_{in}$$

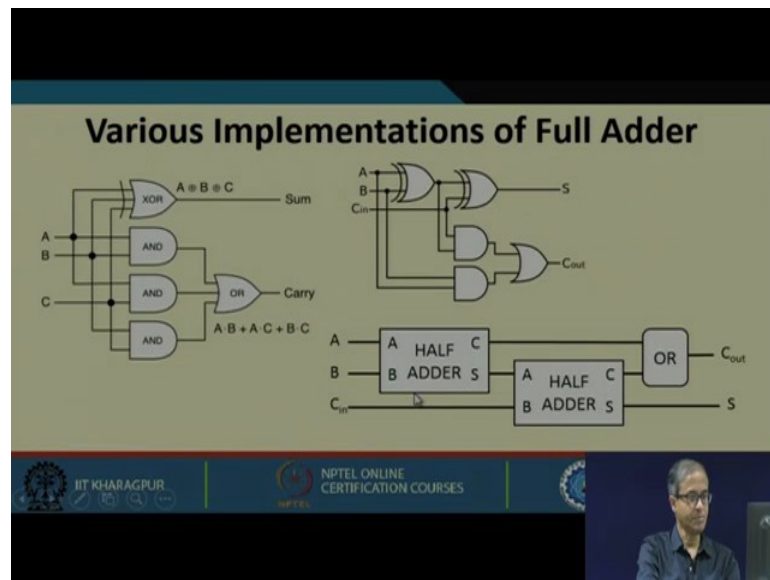
$$= A.B + B.C_{in} + A.C_{in}$$

Logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES are visible at the bottom of the slide.

Let us now see how a full adder looks like. This is the truth table of a full adder where I have 3 input bits A and B and sum and Cout output bits. So, in inputs there can be 8 combinations. So, if it is 0 0 0 sum will be 0 carry will be 0; for 0 0 1 sum will be 1 carry will be 0; same for 0 1 0, but if that two of them are 1's then sum will be 0 and carry will be 1; similarly and for the last case 1 1 1, sum will be 1 carry will also be 1.

A full adder can be conceptually in a block diagram form expressed like these 3 inputs and 2 outputs, and this sum and carry expressions can be written like this. Sum expression if you just consider the output of the sum is 1 these 4 terms and the main terms will be  $A'B'C_{in}$ ,  $A'BC_{in}'$ ,  $AB'C_{in}'$  and  $ABC$ . So, if you do a simplification of these, you will find that this is nothing, but the exclusive or of A B and C ( $A \oplus B \oplus C$ ). Similarly for  $C_{out}$  as the four places where it is 1 it will be  $A'BC_{in}$ ,  $AB'C_{in}$ ,  $ABC_{in}'$  and last one is  $ABC$ . So, this again if you minimize it will be just AB or BC or AC ( $AB+BC+CA$ ). So, these are the final expression for the sum and the carry out .

(Refer Slide Time: 19:15)

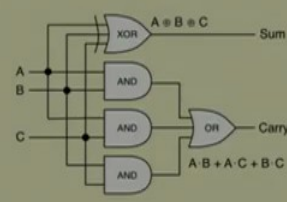


So, when you implement full adder there are various ways of implementing this circuit the first circuit directly implements this function. S is the exclusive or and C is  $AB + BC + AC$ . So, it directly implements the C<sub>in</sub>. C is the C<sub>in</sub>.

So, an exclusive-or to generate the sum, and a 2-level and-or network to generate the carry; now there is some scope for optimization if you break this 3 input XOR into 2 input XOR gates and if you take the output from the first XOR gate to this 2 level and or circuit here also you can generate C<sub>out</sub> and your circuit complexity becomes a little less this is also an alternate design. And the third diagram that I am showing is that if you have half adders as your building blocks you can combine 2 half adders plus AND OR gate to design of full adder these are the alternate designs.

(Refer Slide Time: 20:22)

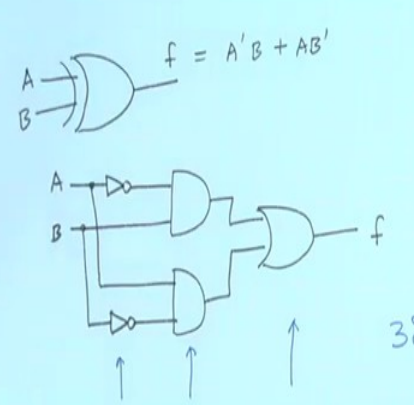
- Delay of a full adder:
  - Assume that the delay of all basic gates (AND, OR, NAND, NOR, NOT) is  $\delta$ .
  - Delay for Carry =  $2\delta$
  - Delay for Sum =  $3\delta$   
(AND-OR delay plus one inverter delay)



IIT KHARAGPUR  
NPTEL ONLINE CERTIFICATION COURSES  
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA


So, let us look at the design like this let us make a simple delay analysis. So, we assume that the delay of the basic gates as 1. So, what are the basic gates? Let us assume AND, OR and NOT are the basic gates.

(Refer Slide Time: 20:47)



$f = A'B + AB'$

$3\delta$



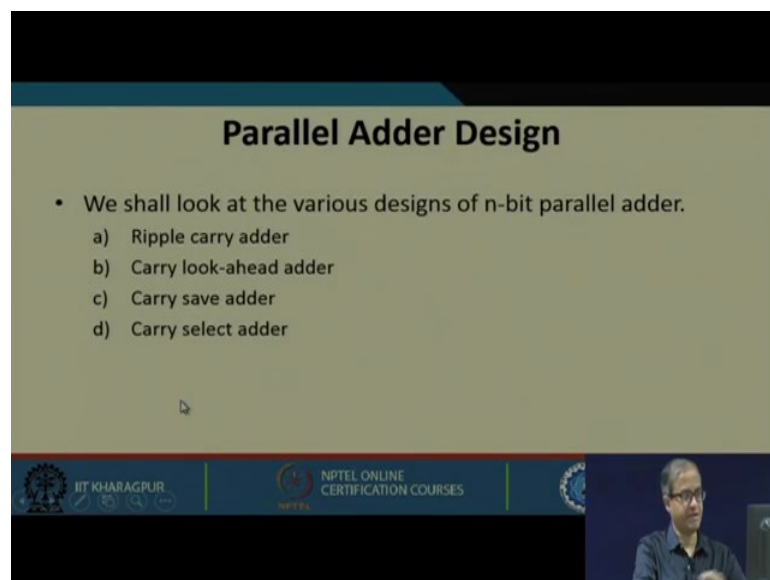
But what about XOR gate let us look at a scenario of a 2 input XOR gate. Let us say I have 2 inputs A and B and an output of f. So, what does f represent  $A'B$  or  $AB'$  bar. So, if I can implement it using 2 level AND OR this will be my f. The first input will be fed with  $A'$ . So, if this is A there will be a NOT gate to generate  $A'$  and then AND with B.

Second one will be  $AB'$ . A will be fed as it is and there will be  $B'$ . So, here if you look from the input to the output you will see that there are 3 level of gates one is to negate the gates then these AND gates and then these OR gate.

So, the equivalent delay of a 2 input XOR will be thrice delta if delta is the delay of a basic gate. This is what is mentioned here that for the XOR gate for this sum the delay will be thrice delta and for the carry this circuit I am already showing here it will be 2 level gate delay.

Student: (Refer Time: 22:18).

(Refer Slide Time: 22:16)



**Parallel Adder Design**

- We shall look at the various designs of n-bit parallel adder.
  - a) Ripple carry adder
  - b) Carry look-ahead adder
  - c) Carry save adder
  - d) Carry select adder

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now let us see how we can design a parallel adder. Parallel adder means we are trying to add more than 1 bits together. For n-bit numbers we have already seen how we can this. The same concept you can extend in the first adder design that we shall be seeing there are several main types of adders that we will be exploring, ripple, carry look ahead, carry save, carry select adders etc.

(Refer Slide Time: 22:56)

### Ripple Carry Adder

- Cascade n full adders to create a n- bit parallel adder.
- Carry output from stage-i propagates as the carry input to stage-(i+1).
- In the worst-case, carry ripples through all the stages.

$$\begin{array}{r} 111110 \leftarrow \text{Carry} \\ 011111 \leftarrow \text{Number A} \\ + 000001 \leftarrow \text{Number B} \\ \hline 100000 \leftarrow \text{Sum S} \end{array}$$

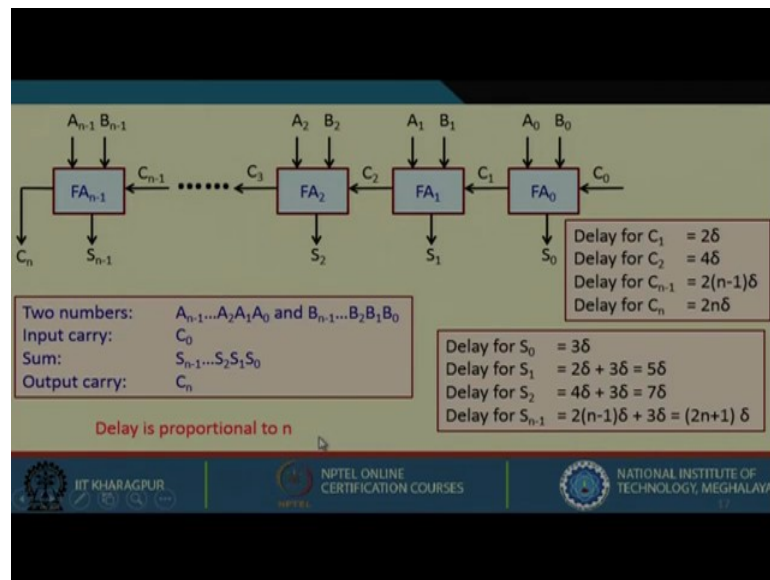
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

But let us look at it 1 by 1; ripple carry adder is the simplest which models the way we do addition using a paper and pencil method, just to recall again you take this example once more. So, the 2 numbers A and B are given we add the least significant digit of the numbers with an initial carry of 0 no carriers there you get a sum.

Student: (Refer Time: 23:24).

You get a carry in the next stage you again add these 3 digits, you get a sum and a carry you again add these 3 digits and this way you repeat. So, if there are n digits in my binary number, we require n such adders.

(Refer Slide Time: 23:48)



Let us see this n- full adder stages will be like this.

Student: (Refer Time: 23:54).

So, my input numbers are fed bitwise in this form is  $A_0, B_0$  to the least significant stage a  $A_1, B_1$  here a  $A_2, B_2$  here and  $A_{n-1}$  and  $B_{n-1}$  will here; and as you can see that the carry out of the first stage the is going as a carry input to the next stage, carry out of this stage is going as a carry input to next stage and so on. This is a parallel adder that I have designed just using a cascade of several full adders. So, I have several full adders and every full adder is capable of adding 3 bits. So, I am adding 3 bits like that and there will be 1 full adder at every stage which will be handling the addition of those 3 bits; and when the addition is carried out the full adder is generating a sum and it will be generating the carry for the next stage.

So, the example that I have shown in the worst case the carry might be propagating from the least significant stage to the most significant stage for ripple carry adder circuit. So, what we assumed is that the two numbers are A and B both are n-bit numbers the input carry is  $C_0$  this is the carry in of the last stage, sum is  $S_0$  to  $S_{n-1}$  for n-bits and there will be a carry out  $C_n$ .

Now if you want to calculate the delay it is fairly simple. See for a full adder we have already calculated the delay, and we have seen that the delay for the carry out is twice



delta; so for the first full adder when I apply  $A_0 B_0$  and  $C_0$ . So, after  $2\delta$   $C_1$  will be generated. So, it is only after  $2\delta$  in the worst case this  $A_1 B_1 C_1$  will be available. So, you wait for another  $2\delta$  you will be getting  $C_2$ . So, for  $C_2$  will be getting at four delta  $C_3$  will be getting at  $6\delta$  and so on.


$C_{n-1}$ ; so if you extend it will be after  $(2n - 1)\delta$ , and the final carryout will be generated after  $2n\delta$ . So, for generating the carries your maximum delay is  $2n\delta$ ; let us now look at this sum. See for sum as you recall it is an XOR gate. So, delay will be  $3\delta$ . So,  $S_0$  will be generated in  $3\delta$ . Let us look at  $S_1$ , this carry  $C_1$  is generated at  $2\delta$  time. So, starting from  $2\delta$  this will require another  $3\delta$ , that will be  $5\delta$ ,  $C_2$  is generated at  $4\delta$  time. So, starting at  $4\delta$  you take another  $3\delta$  for  $FA_2$ . So,  $S_2$  will be generated at  $4\delta + 3\delta = 7\delta$  time. Look at the last stage;  $C_{n-1}$  is generated at this time  $(2n - 1)\delta$  so that plus another  $3\delta$ , which is  $(2n + 1)\delta$ .

So, you see out of these  $(2n + 1)\delta$  is larger. So, the worst-case delay of the ripple carry adder is  $(2n + 1)\delta$ . This grows linearly as the value of  $n$  increases and this is mainly due to the ripple effect of the carry. So, I have already shown the worst-case scenario that when you apply an input carry the worst case the input carry will be rippling through all the stages and to the final output fine.


(Refer Slide Time: 28:09)

### How to Design a Parallel Subtractor?


- Observation:
  - Computing  $A-B$  is the same as adding the 2's complement of  $B$  to  $A$ .
  - 2's complement is equal to 1's complement plus 1.
  - Let  $X_i = B_i'$ .



IIT KHARAGPUR



NPTEL ONLINE  
CERTIFICATION COURSES

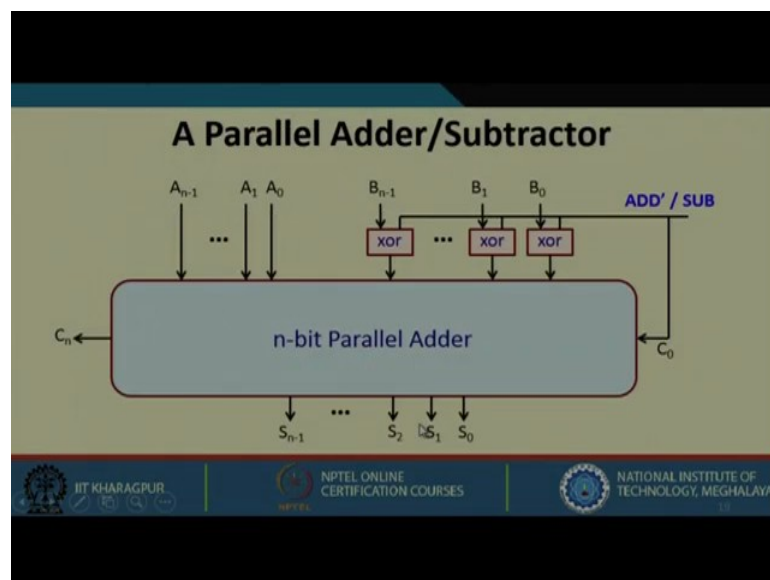


NATIONAL INSTITUTE OF  
TECHNOLOGY, MEGHALAYA

So, this is how a ripple carry adder works; delay is proportional to  $n$ . So, how could design a parallel subtractor, I have already shown you that how to design a parallel adder. Now already we have seen how 2's complement subtraction is done you take this 2's complement of the number you want to subtract and then add it. So, it is fairly simple you see I am just showing you a schematic diagram let us say for every bit of this number  $B$ , let  $x_i$  denote  $B_i'$ . So, what we do we take our normal ripple carry adder just like we have shown?

The first number  $A$  we apply as usual; second number instead of  $B$  you apply this  $x$ . So, what is  $x$ ?  $X$  is essentially the 1's complement of  $B$  --- you are complementing or doing a NOT of all the bits, and another change you are doing, carry input see in the initial cases this carry input initially would be 0, but here we are saying that the carry input is set to 1. So, essentially I am applying 1's complement and adding a carry input of 1 which effectively means I am adding a 2's complement number. I am doing a 1's complement plus another extra 1 that makes it 2's complement. So, what this adder will be computing is nothing, but the sum of  $A$  and the 2's complement of  $B$  right. So, what will be getting in the final sum output will be the final result  $A$  minus  $B$ . So, how you can design a general adder - subtractor?

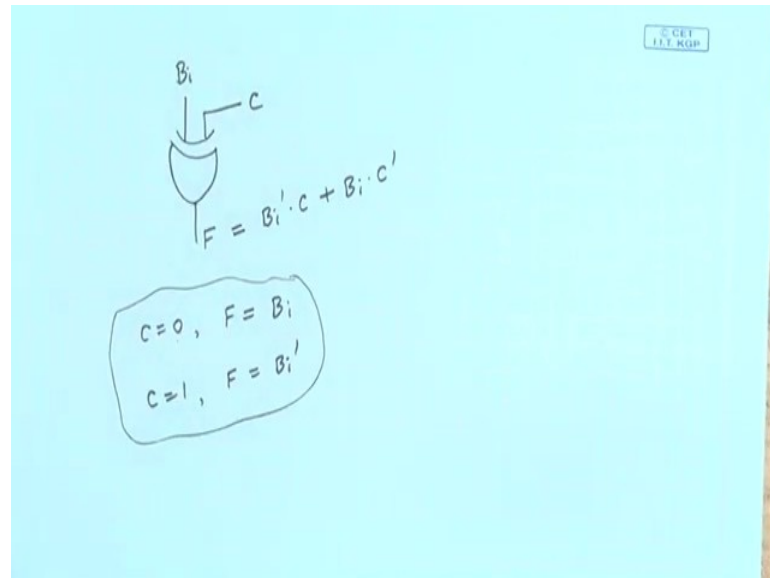
(Refer Slide Time: 29:52)



Then let us have a circuit like this we have a  $n$ -bit adder I apply the first number here  $A_0$  to  $A_{n-1}$ . Well I apply the second number not directly, but through a sequence of XOR

gates. So, how are these XOR gates connected? One of the input to the XOR gate is connected directly from an external control input add/sub, and the other input the second input of this XOR gates are fed with the bits of the second number  $B_0 B_1 B_{n-1}$ .

(Refer Slide Time: 30:40)



Suppose I have an XOR gate. Let us say I have 1 input as let us say  $B_i$ , I have a second input  $C$ . If this is  $F$ , let us say if  $C$  is 0, then you can easily check  $F$  will be same as  $B_i$ ; if  $C$  is 1,  $F$  will be same as  $B_i'$ . Because if you write down the exclusive or function it will be immediately clear you put  $C = 0$ ,  $B_i$  will remain; you put  $C = 1$ ,  $B_i'$  will remain. So, you can use this XOR gate as a controlled inverter; if  $C$  is 0 no inversion, if  $C$  is 1 then there will be NOT. So, essentially here this XOR gates are being used as a controlled inverter, if this control signal is equal to 1 which means the 1's complement of this number  $B$  will be fed into this input of the adder, and this  $C_0$  will also be 1 same input is will fed here.

So, now, we will be doing a subtraction, but if this control input is 0, so the carry input is 0 as well as  $B$  will be directly coming to the input of the adder. So, now, it will be acting as an adder. So, this simple circuit just by the addition of a few exclusive OR gates we can have a combination where you can implement both an adder and a subtractor using this same hardware circuit.

We shall see some adder designs which are faster than the ripple carry adder that we have already seen in our next lecture, it means we have also seen one thing that how we can

combine addition and subtraction. For subtraction we really do not need a separate piece of hardware we can use an adder directly to carry out subtraction as well. So, this was the main topic of our discussion in this lecture.

With this we come to the end of this lecture. As I said in our next lecture we would be looking at some other kind of adder designs, which are in some sense better or faster than the ripple carry adder that we have seen.

Thank you.