

Advanced VLSI Design
Prof. A. N. Chandorkar
Department of Electrical Engineering
Indian Institute of Technology – Bombay

Lecture – 11
Arithmetic Implementation Strategies for VLSI -Part II

Okay, continuing with our effort to design arithmetic data path in the implementation in VLSI, Last time we have seen the general introduction to data paths, we also have seen the requirement of high speed data paths, you also have seen that in current context not only the speed is of relevance, but it is also that we have constraint on the power, so the next few hours of my talk on arithmetic, we shall also first look into the basic structures of data path blocks.

And then go over to say how the design low power or low voltage arithmetic in the current requirement. particularly for 45 nanometers down technologies, though the example may not actually work on particular 45 nanometer models, but we say if scale down the standard CMOS process from say. 130 nanometer down to 45 or below the technique will remain similar and we shall be able to reduce the power anyway the we have discussed.

Last time that there are number codes different kinds of codes can be used for different chip implementations but the simplest of all which allows simple arithmetic to be performed is of two kinds one is in which one is called the bit serial, in the bit serial the data actually appears bit by bit. Even If you have 8-bit data, you require 8 clocks cycle to enter the data and if you have larger number of bits, you will have to that time to enter the bits.

The advantage of bit serial is obvious that since the hardware is single let us say one full set adder if required to add two numbers but you will require 8 bits of 8 times this addition is to be performed, since the hardware is minimum, so you can certainly say it has saving hombra silicon real state and therefore the cost, the second possibility is that you do bit parallel, that means you actually introduce almost all the bits simultaneously into the adders, but obviously since you have 8 bits or 16 bits.

So as many adders will be required for each bit to enter or each combination of bits to enter and because of that the hardware requirement will be very large and since you are parallelly inputting the data it seems to many and actually as I say later, it may not be true, but it is normally one believes that the parallel bit processing will be faster than the serial bit.

(Refer Slide Time: 02:56)

Arithmetic

1. Bit Serial: Less of Hardware, so saving on the Silicon Real state.
2. Bit Parallel: Simpler in implementation and appears to give faster circuits !!

- ▣ In Reality, ratio of speed in the two cases is very 'small' as parallel implementation has long carry propagation paths.
- ▣ Area of 'Bit Parallel' chip will be larger[by word-length(W_d) times]
- ▣ Both are roughly similar in power dissipation.

Therefore the optimum is *Series-Parallel* for chip implementation.

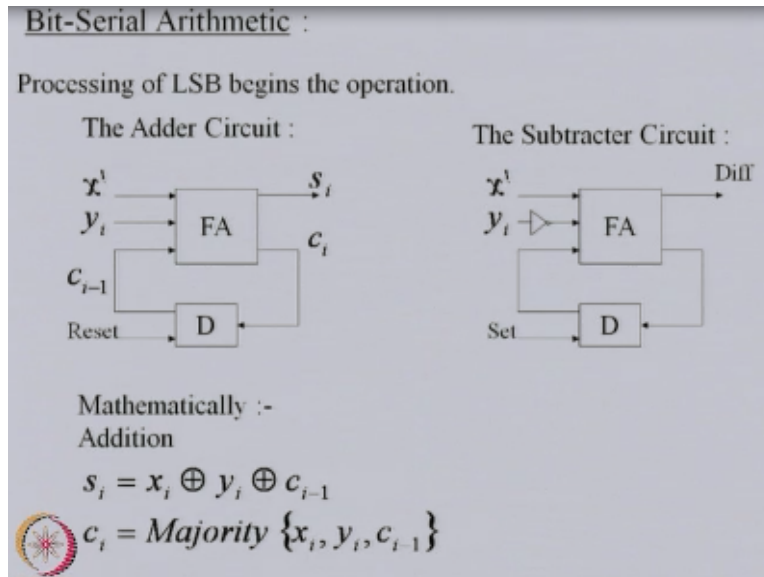
But in reality, if you really do the implementation in VLSI, it is found that the ratio in speed in the two cases is very small as parallel implementation, actually though it looks to be simpler, but it requires a carry propagation and in larger the number of blocks both the bits go through and carry propagation time increases, so in reality one that the speed of a parallel processor or parallel adder or parallel multiplier is not much different.

It is definitely lower than the bit serial processor, but it is not as low as one things obviously of course the area bit parallel chip will be larger because you have to have large as many word length times you have to do it because if you have 8-bit data you require 8 processors or 8 adders or 8 multipliers, so obviously you have a large amount of hardware going on and therefore area is very high compared to bit serial and also it has been found.

Since you are anyway going to do operation even in bit serial 8 times and in bit parallel you are going to do simultaneously 8 times, so in power dissipation case not much is different between the two, how we are looking at two kinds of hardware available or bit serial or bit parallel adders

to multipliers are available, it is a prudent in thinking that if I can do what we call as hybrid kind which is series parallel chip implementation which may be easier to optimize than either of bit serial or bit parallel.

(Refer Slide Time: 04:29)



Typical hardware looks something like this, you have a bit serial arithmetic and bit serial arithmetic is relatively simple, you have a full added circuit shown here, which receives input x_i and y_i and gives output S_i and a carry out which is C_i , now the way it is the first bits are entered first and you generate a carry, and this carry is now required to be fed back for the next input bits $x_1 y_1$ and since you require 1 clock cycle.

To enter the next bits, you also have to now delay the carry through a flip flop to return the input of adder, so that they are actually are in the phase with the next input $x_1 y_1$ and this process continues as long as the input data stream appears at the input of FA now mathematically speaking we will see little later again but the sum could be expressed as $x_i x$ or y_i or $X_i C_i$ minus 1 and C_i which is the carry output is essentially the majority of $X_i Y_i$ and C_i minus 1.

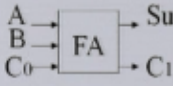
Another interesting circuit many of the operations in the case of arithmetic you require is to subtract the data from one from the other, we know once a complement technique or two complement technique does allow the simple thing to happen, bit even without giving the

complement data you can just have XI YI bar as the input to full adder and by this the output will be then differential difference of the x and y.

So, this subtractor adder circuit are similar except one of the input which you need to subtract from the other one has to be given as an inverted output inverted input, now if you look at the structure of this.

(Refer Slide Time: 06:26)

Bit-Parallel Arithmetic :
 Full adder is the basic element and uses 2C number system.



$S = A \oplus B \oplus C_0$
 $C_1 = A \cdot B + A \cdot C_0 + B \cdot C_0$

Thus, $S_i = A_i \oplus B_i \oplus C_{i-1}$
 $C_i = G_i + P_i C_{i-1}$
 $G_i = A_i \cdot B_i$ Generate signal
 $P_i = A_i \oplus B_i$ Propagate signal

For CMOS implementations, it can be further reduced as:

$$S_i = \overline{C_{i-1}}$$

$$S_i = C_{i-1}$$

$$C_i = C_{i-1}$$

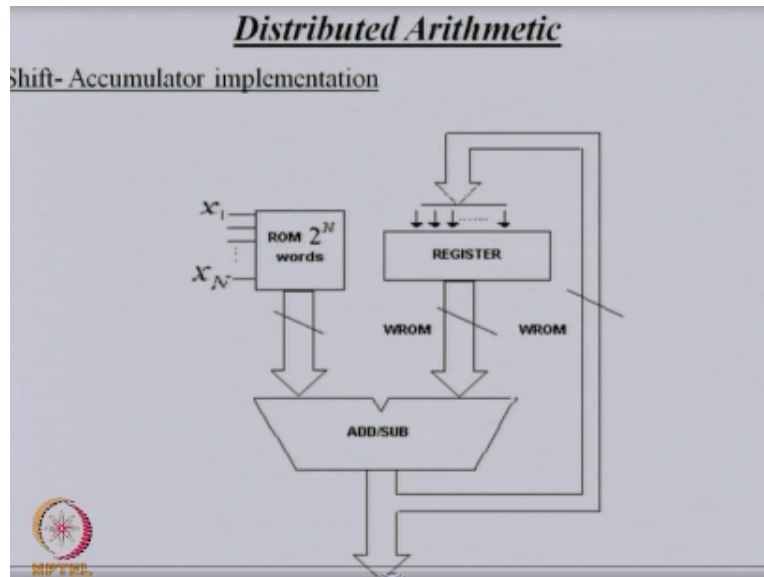
$$C_i = A_i$$

If you look at the bit parallel arithmetic correspondingly one can see from here it is a full adder which is shown similar like this except the part which is not fully shown to you here is that if you have a AI BI and I may be 8 or 16, 20,32 or 64. For each such a0, b0, a1,b1,a2, b2 up to AI BI you will require each of them one full adder and then everyone has to generate a sum and a carry of the first full adder block is to be fed as they carry it to the next full adder block which receives the next bit a1 b1.

So obviously there is carry propagation going on from adder 1 to adder 2 to adder and so on and so forth, typical expressions almost every one of us using is x as X XOR B XOR ci or c0 as the 0 input carry and c1. Please remember generally if it is the first full adder the in out carry is normally 0 because there is no number initially coming from anywhere however if it is in between this c0 will be actually received from the last full adder stage.

The carry part is essentially as a r of three and gates a.b, b, co will express them in some other form, little later once again though I right now wrote itself that Si Ci I can be explained what in terms of we generate and propagate signal but let us wait, for we come back to explain this.

(Refer Slide Time: 08:04)



The other possible arithmetic the other method of actually implementing a logic or arithmetic adder or multiplier is essentially done through a distributed arithmetic which essentially is what used in almost every processor, your input data is actually loaded on the ROM, which if you say in bit data is 2 to the power of n words around ROMD capacity each such first bits are transferred to adder.

And initially there is no carry which is coming from there then the first sum will be actually outputted, and a carry will be fed back this is essentially 8 bits or 16 minutes registered each carry it fed here in 8 bits or 16 bits and that register after a clock may transfer through another ROM which is storing the data and will fed back to the new data which is appearing and this process of feedback.

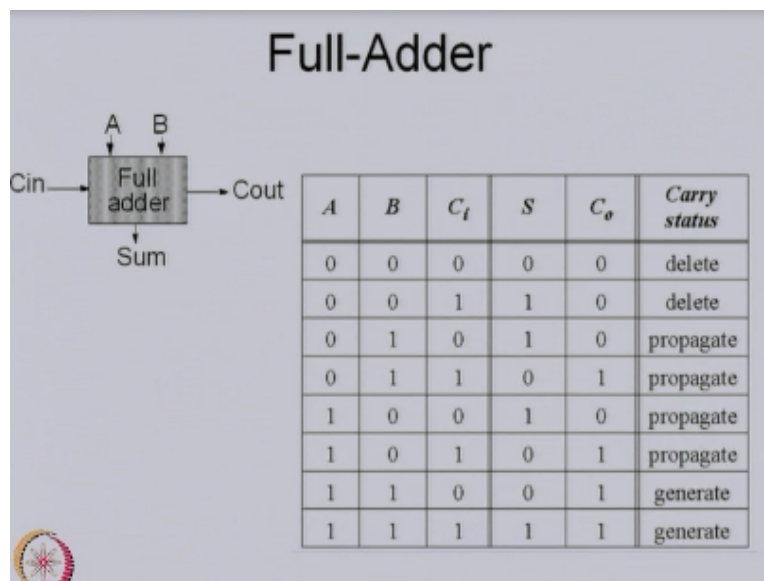
Which is essentially what I say with parallel and this is how the data can enter this can be bit parallel and this is the add or subtract block which may be a single block or may be a parallel block or may be combination of the two, and since you are actually putting register and to shift

that and also you have the memory to store it and we also have the shift accumulator implementations.

This is what typically we do in real life, so we start with the adders in specific now and we like to see how it is the advantage of actually using hardware, what kind of logic style we use for adders and please remember from the processor. of view we keep saying that the system should be more programmable, whether in the case of VLSI we are more interested to see the operations are faster.

That is speed is higher and we are also interested that consume much less power than what one thinks and probably possible even the area be minimized so there are three parameters in VLSI. We optimize the power, the speed and the area. Of course, we may not optimize all three together, but these are the steps on which I will actually test or benchmark every circuit. We will think about and check whether in a specific requirement which adder can be used.

(Refer Slide Time: 10:29)



Here is a typical truth table for a full adder circuit. Shown here, you have a full adder, you receive an input A and B and you have an input carry and you have an output carry and the sum. And this gives to the table. And you know this is very interesting. This is called carry status. We will look into this, how we are going to implement the hardware. If your A and B be the data input 0 0 and the carry is initially 0, we know that sum is 0. And then you do not need any carry.

And this happens as long as one of the bits become 1. The sum cannot become 1 because $1+0+0$. It only occurs when one of the bits of the three becomes one. However, if there are 3 bits becoming one, adding them the sum will go to $0+1+0$. But it will then generate a carry and by same logic we keep saying this. Now interesting feature which I will come back again, once again when I come back to this word DPG.

Whatever I am talking here, this is essentially other terms, it is called kill, this is called generate and this is called propagate. So, one can see very carefully that if A and B are same, okay let A is 0 and B is 0, if CI is 0 or CI is 1, the carry still remains 0. Okay so essentially, I am trying to say that irrespective whether carry is 0 or 1, as long as A is 0 and B is 0, the carry is not needed. It will always be deleted.

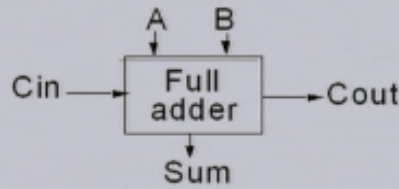
That is, whatever carry it has initially, it will become 0 or if you look at the last 2 where again A and B are one, that is equal, but they are one now. The carry will always be required as 1. So, we say that you need to generate a carry which issues a delete means 0, kill means it is going to be 0, and generate means that you have to create output as one for the carry. In between if you see, a very interesting thing you can see independent of what A and B, you can see in between.

As long as they are not together 1, these four cases 01111010 one can see from here that if CI is 0, Co is 0, CI is 1, Co is 1, CI is 0, Co is 0, CI is 1. So, what is essentially happening is that, Co is same as CI or to say in other terms, CI propagates. And therefore, having seen this table, I think one can get a good idea what kind of circuits we should implement. Which for given a data, we will do the following operations.

Of course, this is essentially arithmetic and therefore can always be represented in arithmetic form.

(Refer Slide Time: 13:24)

The Binary Adder



$$S = A \oplus B \oplus C_i$$

$$= A\bar{B}\bar{C}_i + \bar{A}B\bar{C}_i + \bar{A}\bar{B}C_i + ABC_i$$

$$C_o = AB + BC_i + AC_i$$

So typically, a binary adder shown here, which is what the expressions will show, $A \oplus B \oplus C_i$ is the sum which is coming up as A and B and if you expand XORs, you will get 3. This is sum sub product. Okay and the output carry will be essentially sum of the 2 bit terms of this, which contains 2 C_i terms and 1 AB terms, and if you have larger numbers ABCD then you will have larger sums and carry terms available.

Now before we go ahead, I would like you to see a very simple implementation of whatever I said so far.

(Refer Slide Time: 14:07)

LOGIC REPRESENTATION OF
ADDER

2 Bit Adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

with Initial carry C_{in}

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

$$C_{out} = (A \oplus B) \cdot C_{in} + A \cdot B$$

Full Adder

Okay I have a logic representation of an adder which is a 2-bit adder. A is equal to B. Yes, I have A B S and C s my A and B are input, s is sum, and c is carry. And the basic idea as I say is s is $A \oplus B \oplus C$ and, so we have 2 ways of doing it. We do partial addition or what we call half adder. And if you have a half adder that means you only do $A \oplus B$ as assuming that the input carry is not existing.

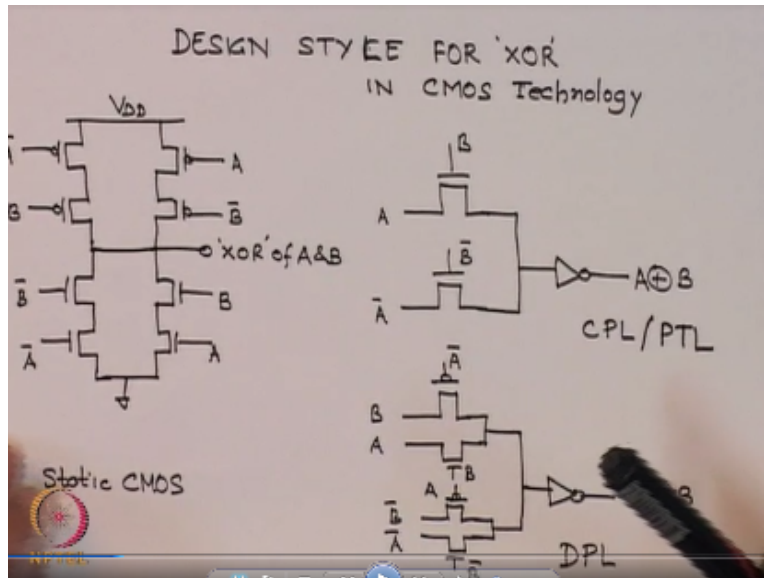
In that case we only have to sum without carry inputs. So, we have summing in nothing but $A \oplus B$. So, a simple circuit of XOR which receives two inputs A and B can give you sum part and a carry Of course this is then the AND gate of A and B, which is nothing but A . B. However if you have an input carry then you have to do 2 XOR. Now $A \oplus B$ is one function then you have to XOR it again with C_n .

As I have shown here $A \oplus B \oplus C_n$ will give me the sum but the scene can be expressed in simple other form, $A \oplus B \oplus C_n$. Since $A \oplus B$ has already been created by me I can put an AND gate with C_n as another input as one of the term in C out. The second term is nothing but A.B. So, on another AND gate A.B, all of this two will give me the carry. So, if you see, very simple way is you see your two AND gates, 1 r gate, 2 XOR gates can implement the full adder terms in real life.

Now why we are interested to look into this, the reason why I am interested in that is, how do I implement them in real circuitry. Assuming that we are continuing to use only CMOS blocks, you can use Bi CMOS, you can use even Bipolar. But as I say, the current technology may remain for many years, it is the complementary mass technology. So, I talk about the implementation of a simple adder on a circuitry, on a standard CMOS.

As we have seen in our first block, that the major component in a full adder or anything is 2 XOR gates and then all of course is the easiest to implement as we know in XOR in the CMOS.

(Refer Slide Time: 16:39)



So, I only show you the major block in the full adder part in the XOR and I have given you some different styles in which one can implement this. The first one is static CMOS. That means for each N channel device, there is a P channel device. And the way I implement it is that, I have 2 P channel devices, or two series P channel devices receives input A BAR and B. And parallel to them or another 2 P channel devices in series which gets this is, input A and B.

Please remember, these are just complements B, B BAR, A, A BAR. Please remember these are cross because in XOR, you need a B BAR + A BAR B. So, this is what essentially, we are trying. If you look at the N channel part, static part for that, you have A BAR B BAR as the series combination of transistors in series and then they are parallel. And one can see this 8-bit transistor circuit can implement a simple 2 bit XOR.

Now as I say it is static and we may like to compare them with the other style and verify for the VLSI implementations which one of this style, one should use. Please remember for larger bits of size, for a larger speed or a low power, you may have to choose among them as per your requirements. But let us see optimally which one is better. The other possible option which is shown here is, you can see.

I already said you have 8 transistors here for a static for an XOR and an equivalent XOR block can be implemented by using, what we call the complementary pass logic or just pass transistor

logic. Basic idea is simple. It is we are implementing a multiplexer and we know a multiplexer can implement any function. And since we are implementing $AB + A \bar{B} B \bar{B}$, bar of that is XOR. Please remember that the XOR can be mathematically written as, $A \oplus B = AB + A \bar{B} B \bar{B}$.

Essentially this is an XOR but the bar of that will become XOR. So, if I want to implement this whole XOR, all that I do is I use 2 pass gates which it receives as an input to pass gates input A and A BAR and they are controlled by B and B BAR. Please remember A B and A BAR B BAR in the same sense for the same transistor. Now since this is a MUX, obviously B is the 1. Only then A passes. If B is 0, then it does not pass. But B is 1, B BAR is 0. So A BAR does not pass.

If B is 0, B BAR is 1, A BAR passes. So conditionally saying AB is passing here and A BAR B passing here, and this is called wired. Okay and once we do this then invert of this can be actually $A + B \bar{B}$. Now the problem with all passed on transistor logic are called CPL logic since the transistor does not have any power supply. Of course, you need an inverter here otherwise for the function implementation.

But this requires larger size itself because if you have to drive this line ahead then you require power to be given from somewhere. And this inverter will actually provide power through VDD. There is also a problem in this kind of this. Please remember there are capacitances is associated with these nodes. So, when B becomes one let us say and A has to pass one initially here.

So obviously this one to transfer in charge this capacitor will take some time and the larger the capacitor a larger will be the time. Also, one can see from here since it is passed on this step when, let us say I have an input which goes from 0 to one volt or 0 to let us say I have this transistor has VDD is let us say 2.1 volt and V_T is 0.8 volt. Example no problems. So, when input goes from 0 to 2.1 or goes from 2.1 to 1 this and B let us say goes from 0 to 0.21 fully.

So initially when this was 1 that is 2.1 volt and you have 0 volt here. Till 0.8 it does not pass because the transistor I mean 0 1 minus this when it reaches around 2.1, please remember if this is 2.1 volt and you are going to 2.1 volt so till 0 it will pass because this V_T is much larger than

V_T . But when you reach $2.1 - 0.8$ which is roughly 1.3 volt then a little ahead of that then you can see $V_J - V_T$ is less than V_T $V_J - V_T$ is smaller than this, so the transistor turns off.

Therefore, the maximum output of this is 1.3 volt even if the input was at 2.1 volt. So, there is what we call one threshold drop across a pass transistor. So, with CPL logic problem is you do not reach the full signal and to restore them you need an inverter. So, you need a power source here and you need a restorer of the level which essentially come from inverter. In the case to avoid some these problem that is an interesting circuit has been suggested this is called double pass transistor Logic

Which elevates of course the N channel transistor shown here also had two more problems one is feed through due to this capacitance here the data output is always connected to the input which is therefore changes the $V_J - V_T$ on this and because of that there is an issue of charge sharing goes on. This is called feed forward issues. Charge sharing, and feed forward are the two major worries is a pass gate.

Third of course what we saw kT/C noise which cannot be minimized. However, if we want to reduce this feed through and charge sharing problems there are other methods which are suggested. One of course is you can directly put a N channel P channel transistor and put by bar and similarly across you put with n channel P channel transistor. But that essentially does not give you full control.

So, the modified CMOS pass transistor logic or complementary logic has been called double pass transistors logic which is shown here which is very simple. Here, we have 1 P channel transistor and parallel N channel and 1 P channel transistor which has N channel parallel to it, but they do not receive same inputs that is an interesting part. First one which is P channel input which is B, second one which is N channel input which is driven by controlled by B input A.

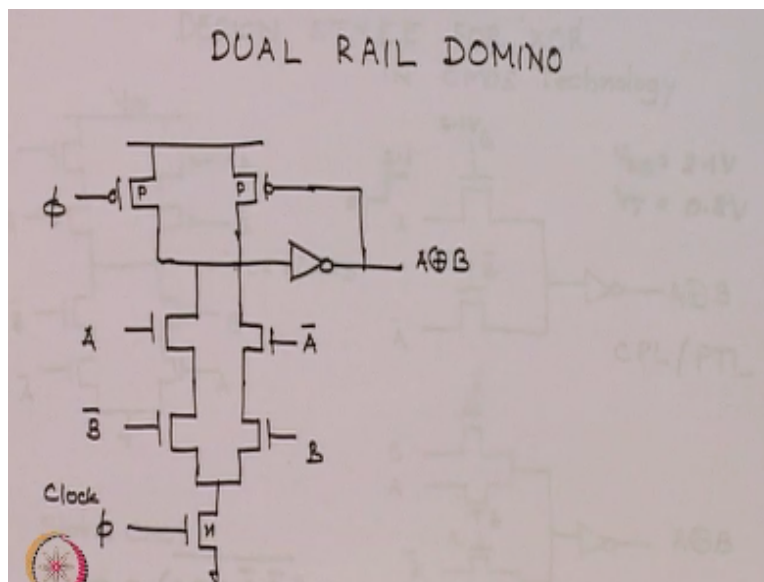
And the same now is opposite complement that this is \bar{A} and so this is A this is B therefore this is \bar{B} this is B so this is \bar{B} the complement of that, as we did complement here, the

same is complemented here. Now in this case in either case the level restorer is already there but then since one of the transistor will be on for a while 0 will be passed through this and VDD will be passed through the other one.

And therefore, at any time these levels will reach here will not have VT drops also since this is a CMOS circuit or some kind of a complementary part in parallel the charge sharing can be now done through either this or this, this loss of charge can be provided by one to other therefore there is no charge sharing real problems. Feed forward is a problem but you can see it is a compensated problem and therefore much less speed through problems appear.

So, pass double pass transistor logic which is actually very interesting logic can also implement on X which is shown here. Please remember in this case you have 2 transistors for A B bar and A bar and B bar and 2 inverter transistors so 4 transistor we have 8 here four here. In this 2 + 2 + 2 which is 6 transistors, so you have 4 transistors 6 transistors and 8 transistors. So, you can see clearly see the area of this is smallest next is this and the largest in this.

(Refer Slide Time: 26:03)



There is another dynamic possibility, dynamic logic can be implemented and in dynamic logic the most famous logic we know is the domino. This is called Dual rail Domino. There is a possibility some books think of single but I am just showing you the one which is most logical and which is most easy to pick up which is called dual rail domino.

Now in the Dual rail domino basically what we are doing in a domino we know the logic is passed on it is you know domino has two stages one is called pre charge mode and other one is called evaluation mode. The evaluation is essential on the N channel logic whichever you want to implement and the pre recharge is during evaluation the ground must be provided by the lower N channel transistor during pre charge this is off our P channel should pre charge the output

This is what we are looking for. Now this dual rail domino which is shown here, this is the inverter and I am feeding it back a P channel called VP. The reason why we do this let us say we are pre charging and phi 0 P channel is actually turning on now this becomes 1 so this becomes 0 and this turns on and this essentially is trying to bring this level full because whatever next time when you actually evaluate and when Phi goes to 0, please remember and phi goes to 1 and this will become 0.

This will become 1 will switch off since it is off week size this will leak some and will try to keep capacitor charge even then. so this is a very standard technique and as you see how many transistors I need 2+2 4+2 6,7 + 2 nine transistors. So, area wise this is very very larger than any static also. However, when we came out with Domino obviously we know that it has not by it is a dynamic circuit.

One of the major feature it allows you at no time when PN are fully on or even sharing the current and therefore the static current is very low in the case of dual rail domino or what we call the leakage current can be minimized. In the case of dual domino particularly 45 nanometer down technologies now leakage is a major worry so probably we may have to deal this dual rail domino once again with much more sincerity with what we have done so far.

(Refer Slide Time: 29:09)

COMPARISON OF DESIGN STYLES
FOR ADDER CIRCUIT

Parameter	Static CMOS	CPL	DPL	Dual Rail Domino
Power (mW)	34.3	34.5	27.5	82.5
Worst Case Delay (ns)	2.33	2.24	1.98	1.78
Energy (pJ)	79.9	77.3	54.5	146.9
Area: in Total Transistor Width (μm)	27355	17437	19117	32444

Now we can do some comparisons of different styles. This is a standard 1.2-micron technology magic was used for layouts and the standard spice 2 g was actually used to stimulate this different blocks which are shown here. With these kind of technologies with supply of 3.3 and the frequency of chosen is 100 megahertz. We can see from here these tables gives me the parameter of my interest in VLSI and these are different styles which I can implement in XOR.

This is still XOR and hand based total added circuit requirements and the circuits are shown only for example but similar thing you can do for Nans and nons or nors and can therefore this is total added circuit parameters evaluation. So, if you look at power delay or worst case delay energy which is power into delay and area which is the area is normally expressed in widths microns simply because W into L , L is common to all of them.

So, you can see $W_1 + W_2 + W_N$ into L is the total area. We always expressed per unit length upon unit channel length and therefore generally the bits are actually expressed total width requirement is what we actually talked about in area, please remember area will be $w_1L + W_2L_s + W_{as}$ any transistors in series or parallel you have to calculate that, and the net w is essentially w into L net w into L is the area, so we only express in W because L is constant for a technology.

So, for example if you look at static the power is 34.3 milli watts. The worst case delay is too 2.33 server energy is 79.9, now please remember the worst word comes from two reasons. One is

for a given input data. If the current required is larger for a given data then use it will consume lower higher power but or its in it is some kind of a critical path, where the current available is smaller.

So, it will charge slowest, so for an input different input data whichever is the slowest, is called the worst case delay and that has been specified here. So, I repeat, power is 34.3 average powers. This is 2.33 Nano second is the delay energy is 79.9 micro jewels an area width is proportional with is 27355 microns. If you look at the CPL or PTL the power is 34.5, delay is 2.2 for Nano, second energy 77.3 micro jewels in a TI is 17 for 17,000,437 microns.

If you look at the double pass transistor logic, the power is 27.5, delay is 1.98, energy is 54.5, area is 19117 and if you compare it with the dual rail Domino the power is 82.5 microwatt milli watts, worst-case delays 1.78 nanosecond, the energy is 146.9 and area is 32445 largest number of transistors. So obviously you can see larger the number of transistor, larger is the area smaller the number of transistors, smaller will be the area.

Okay now, one can see from here. If you carefully look at this table, if you are only looking for high speeds, only looking for high-speed high-performance, obviously the dual rail Domino has the highest speed, but if you look at the power requirement is excessively high 82.5 milli watts. Correspondingly if you see just the energy product of the two is 146.9 micro jewels, whereas if you look at the static comparison wise it has a power is 34.3 which is much smaller than dual rail.

It has a delay which is larger than 1.78, a 2.33 nanosecond but the energy is just around 80 micro Jewel compared to 146 an area of course is slightly smaller because it is 89 transistors in Per cell we saw, so it is 27,000 compared to 32,000, so if in general you can see both of them both as well as speed, as well as on the area, and also on power are not very suitable comparatively yes. But on power scale if you only look you have a CPL which is 34.5 and you are DPL which is 27.5

Correspondingly delay, and if you say the delay is of course lower than both, start again base 2. 24 but this is even faster. It is 1.98, if you look at the energy therefore is 77.3 which is close to static much smaller than dual, but DPL has 54.5 micro jewels and the area between past and CPL and DPL is not very different, which is hardly few hundred a few thousand. Two thousand micron extra or less than that.

So, what is essentially seen from this unless and of course the speed is the only criteria for you, which is the high performance circuit probably you may use adders which are of using dominance or dual rail dominance, in case about that the cost of power, which is obvious in case you are looking for optimal design to a great extent which is what most circuit designers will prefer then either use a CPL or use a DPL.

And comparison between the two, you can see from here this is slight smaller area just slightly larger area, but then you get much better speeds, much lower power comparatively and therefore the double pass transistor logic is always chosen as a candidate, for optimal adder design requirements.

So, I already said one of the criteria of choice, of any hardware comes from the fact that, you may have a different architectures of adders but each block in that will have certainly excel, and, or gates so which ones to use, which style to use, we already compared them. Now by numbers and we feel that it is always possible to optimize, whatever criteria one is putting for a VLSI chip.

Coming back to what I was discussing, I think why I came back why I did in between. I thought that before I go to the actual architectures; let me tell you that any architecture can be built around one of those design styles. Of course, there are a few more styles which I did not look into one is called Beijing zippers. Zippers are advantages, it is slightly better than dynamic or dominant logic, but then it requires two power supply lines which may require little extra area to run everywhere.

And They are also at times, there is the issue of levels not reaching fully, because of the clock is driving it and at that time you have to require further restorer, which may consume power. However, let us look back to more architectural view. of an adder. So how do we actually implement an adder. In normal case we say, we actually now have three kinds of function.

(Refer Slide Time: 36:45)

**Express Sum and Carry as a function of
P, G, D**

Define 3 new variable which ONLY depend on A, B

Generate (G) = AB

Propagate (P) = A ⊕ B

Delete = $\overline{A} \overline{B}$

$$C_o(G, P) = G + PC_i$$

$$S(G, P) = P \oplus C_i$$

Can also derive expressions for S and C_o based on D and P

Note that we will be sometimes using an alternate definition for
Propagate (P) = A + B

We actually represent first for any inputs A and B. Please remember a and b only two bits in the sense that, it is AI BI A0 B0 A1 B1 and that can be 8 bit 16 bit 32 bit 64 bit any numbers . So, if though I have not written, I specifically but you assume that this is each bit of that large numbers are being represented here, we define three functions. This case one we call a function called generate written as G.

Sometimes some books write small g I had written capital and the capital G or small g, whatever it is written is A and B, so it is generate function, is nothing but AB, then there is another function we want to define, and that definition is propagate function, and we say propagate function is nothing but A XOR B there are certain alternate definitions and they are certain as alternate implementation in which propagate function was defined as A + B.

For our case as of now we will define A XOR B and then there is a delete function sometimes called kill function in some of the hardware which I copied from journals and papers. They may have a date signal called kill, which is essentially on is same as delete which is nothing, but A

bar, B bar. So now what we are trying to say that if you generate this function for A and B which, you can because if I know A and B which is your input data I should be able to know G, I should be able to know P.

Because $A \oplus B$ is known AB is known, so is I have a kill function or delete function which is $A \bar{B}$ dot $B \bar{A}$. So, these can always be simply valued by a simple block, by the design style, which I already explained. So, I can generate GI can have propagate function $A \oplus B$ or P and I can have a kill function or delete function $A \bar{B} B \bar{A}$. So, once I do this, then I know the output carry which is nothing, but a function of GP.

Please remember, what was an output function C_0 is that the $A \oplus B$ into $A \oplus B$ is what we said about. So same thing is it is a function of both generate function as well as the propagate. Please remember it is Carry $A \oplus B$ into $C_1 + ABC_1$ is the function $ABAC_1$ is the other function, three terms you become there. Now they can be represented by one simple function, which is $G + P$ times C_1 .

So, $AB +$ please remember I said at $AB + A \oplus B$ is the C_0 . So, this can be represented as $G + PC_1$. We know sum, sum is nothing, but $A \oplus B \oplus C_1$. So, P is nothing but $A \oplus B$ so One can write sum function as $P \oplus C_1$. See I so now I know that if C_0 GP is $G + PC_1$. And since I know my functions G, I know my function P and I if I need D of course is not needed shown here, but if I need D, I have this function is also known to me.

So, we can always say based on knowledge of data, I can I know this and therefore I actually know what will be my sum and C_1 . This is the table which I shown you earlier. Okay you can see from here, very interesting thing before I go ahead if A and B are 0, Go, which means C_0 only PC if G_0 A and B is 0 independent of what CR, you have if A and B are 0 this term is 0, so output carry is nothing, but P times C_1 if A and B are 0 $A \oplus B$ is always 0. Okay so P_0 okay.

So whatever CR you have the output carry must go to 0. Okay that is what the kill that means, initially whatever carry you had on the output, it must become 0 and if you see this table, the first two terms A and B are 0 whatever is the carry input 0 1 the output carry is 0 that is why I call it

delete. Now if you look at this function once again sum is nothing, but $P \text{ XOR } CI$. Now one can see from here that if $A \text{ XOR } B$ for the input I say is 0 and b 0 then the that is 0 A_0 and B_0 .

So the P is 0 since P is 0 the sum will be $0 \text{ XOR } CI$ if CI is 0 output is 0 CI is 1 output is 1. So, if you look at this it depends on $A_0 B_0$ if C is 0 output at 0. C output is 1. Ok so this truth table which we wrote is essentially representing the function which I wrote here. Look for the other one which is more interesting propagate function in the propagate function A and B are either 0 or 1. So if A is 1 and B is 0 is 0 and B is 1 G is still 0.

So, this term is still 0 $A \text{ XOR } B$ certainly always B_1 1 because one of them was 1 and not both. So, P is one so if this is 0 P is 1. So, what we are saying C_0 is same as CI . Okay, so if you look at this table again for this case, when A 0 1 0 1 1 0 whatever is CI here in the four terms you can see C_0 is exactly same. So, what we say input carry propagates if A is 0 and B is 1 or A is 1 and B is 0. Whatever is your input carry actually propagates to the output carry.

If you look at the sum part since P is 1. Okay so $1 \text{ XOR } CI$ is now decided because one if CI is 0 the sum is 1, if CI is 1 the sum is 0. So, we can see here if CI is 1 the sum is 0, if CI is 0 sum is one. Okay so what is essentially saying that the truth table is much easier represented, much easierly made circuit if you can generate this function independently or maybe last one we can see if A and B are 1. So, this is 1 so G is 1. Okay A and B is one means P_0 .

So this term goes away so C_0 is nothing but 1. So, you can see from here, if A and B are 1 C_0 is 1 irrespective what CI you are because P_0 . So, P I CI will go away. So A and B both one will require your output carry to be 1 and therefore we say you have to generate 1. Therefore, output carry is generated and that is what I was saying C_0 G if P is 0, if you see the same function, if P is 0 then you can say P_0 and CI s, when if it is both are 0, output is 0, if both one of the CI is 1 the output is 1.

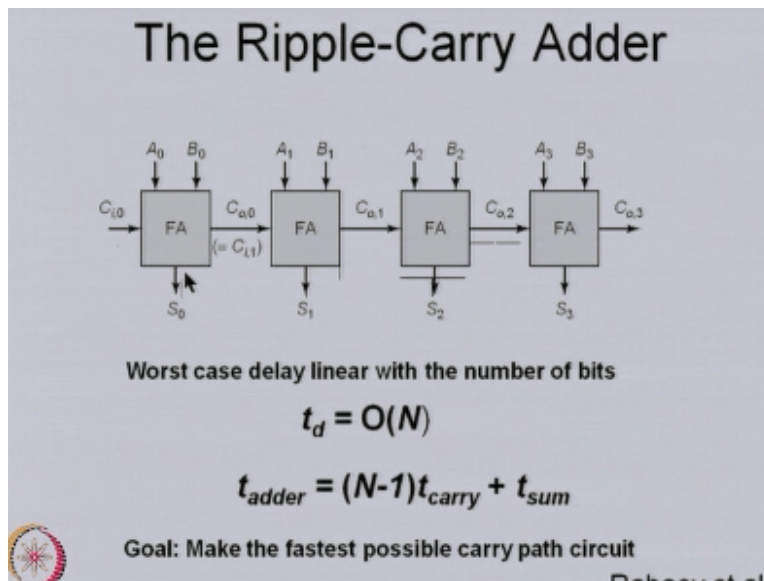
So, if CI is 0 sum is 0, CI is 1. Sum is 1, so the truth table of a full adder essentially can be represented by this status carry status method, and this actually represents the full adder all other implementations ahead, except maybe initial few of them, which is ripple carry or kind will

actually how logic in which we will actually get these PG and these functions and once we get those functions snc can be represented in terms of PNG.

So, this is the technique which all other implementers use. So, you can see why people are looking for this, please remember what is the advantage if something has to remain 0, you do not have to do anything, if it was 0 and it has to remain 0, you do not do anything. If it has 1, you want to retain 1. You do not have to do so; operations can be minimized depending on the function you are choosing the kind of data you receive.

You know which functions will be useful, which functions may not be required, so the hardware can be actually programmed to do is basic requirement, which have lesser power dissipation and lesser area. Okay so the first among them, which is the most common adder which you use, is essentially given by the term our name, which is called the ripple carry.

(Refer Slide Time: 46:27)



Now word itself is saying, carry is rippling so it is essentially a simple task which we are not even generating P and G here. It is independent of any function generation you are a full adder circuits, and we already have discussed that a full adder circuit some is nothing but ax or VX or CI minus 1 or CI 0 and continue like this and some carry of course is A XOR BCN + a dot B or a dot B

So, this full adder can give you two terms A and C, so I give the first input carry I give my first these are shown here, a 4-bit full adder circuit each is this is to bit adder but is a 4-bit adder. Actually, now each first two bits are here, next offers LSB is here and MSPs here but when I do some for LSB I will generate the first some 0 LSB of some, but I will create the carry now and this carry I know is required for summation of A and B.

So even if I impress A and B simultaneously, this carry is not known to me first, when this is operation is over, then I receive this carry then I do this operation, even if this data was known I had to settle this and only then I get this and then I generate this once. I get this with this third bit I can do this and the further fourth bit, when I generates you to then, only I can generate the final carry.

So, one can see from here, if you are a larger number of bit this is only a four bit, if you are 8 bits your carry has to go through like, look at this how many is it three stages, you have to go through, if you are 8 you are seven stages to go through from the carry and first to generate itself, first time one carry and then keep generating a head for and propagated up.

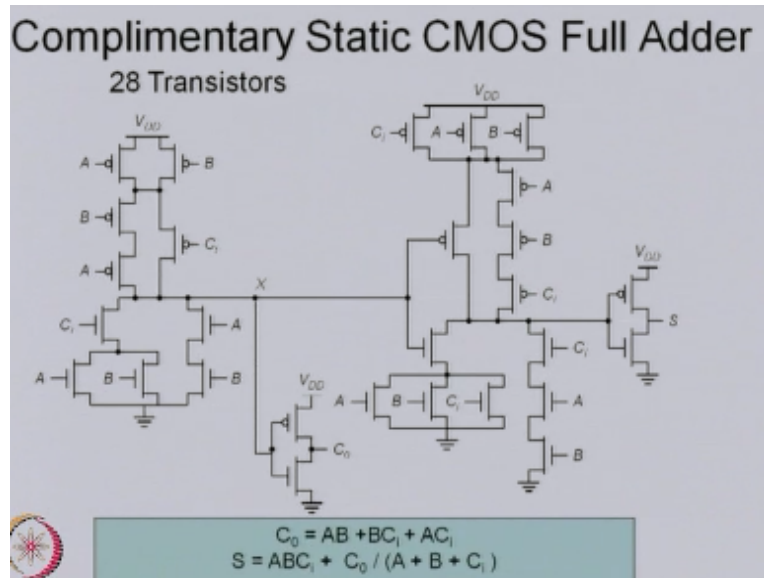
Now this means larger the number of bits, larger will be the delay because carry will require so much, so many additional paths to go through. So typically, one can say I already said 3 bit delay for 4 bits, because only one two three carry propagation. So, the total time for an adder delay which you can get is $N - 1$ times the carry.

Please remember every time because except for the last sum rest are anyway are coming in the same time and this were propagating I got this, and this while propagating I already got this, only this sum, will require last sum carry, because this additional sum, you are not doing because during the carry are going ahead. You are performing the other cases, so one can see that net adder a time is $n - 1 T_{\text{carry}} + 1 T_{\text{sum}}$ which is called the critical time.

The goal of course is to make the fastest possible carry path circuit, this is ripple carry which will be slowest and will be has the delay, please remember if N is very large, then sometime maybe

very small, it is the carry time which will essentially decide the speed of this adder. I just said you are last time I only showed you the XOR circuit, I thought before I leave this part.

(Refer Slide Time: 49:48)



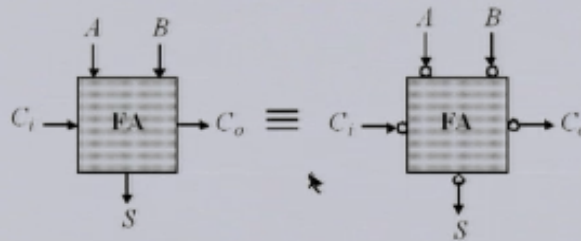
Maybe I will show you the full adder which is implemented here that means I am implementing all of this sum function and carry function, for ABC input. This is my XOR all as you can see from here. Okay and then these other and odd dates whichever is required for Carry and sum are essentially created here. And if you see specifically, some carry of course is very easy to create.

So, it comes very fast but the sum requires all such operations to come each adder part full adder part requires, if you count this in a static CMOS 28 transistors. So, it is a huge area requirement and one of course. One please remember all said and done at the end of the day, I may say implement a possible only on static CMOS because there are other powers which we discussed in our power of your design, which is at very high frequency the issue, is of glitch.

And the leakage power and glitch powers are extremely high and if you have a series connection like this the leakage currents are the minimum and therefore they will be the lower power dissipating circuits at the cost of area which does not seem to obvious when I show you this kind of block. Before I quit this I just want to show the interesting property which logic allows me.

(Refer Slide Time: 51:25)

Inversion Property



$$\bar{S}(A, B, C_i) = S(\bar{A}, \bar{B}, \bar{C}_i)$$

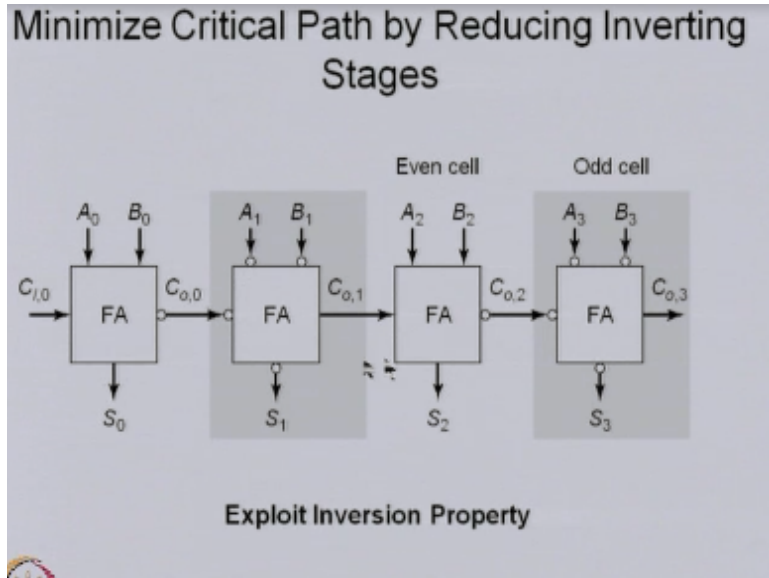
$$\bar{C}_o(A, B, C_i) = C_o(\bar{A}, \bar{B}, \bar{C}_i)$$

If I have a full adder and receive an input in A and B and carry CI. I generate a sum and generate a carry and this is identical which I the full adder which receives the complements of inputs A Bar B bar. I also complements of input carry then I get and if I complement the outputs once again then I get sum and C 0. This can be proved by the simple logic. I repeat you complement all you get complement again back to this.

So, this is very interesting is called inversion property. So, you do not have to do twice such functions. You can have buy an inverter itself you can then you can create. Normally you will always have A A Bar B B Bar C C bar available to you at the end of the day. Every logic will start with its complements and true end complements and if they are already available you can actually use it to your advantage.

So, before we go we can say minimize critical path by reducing inverting stages. Please remember inverting delay is the largest and the non-inverting delay is the smallest.

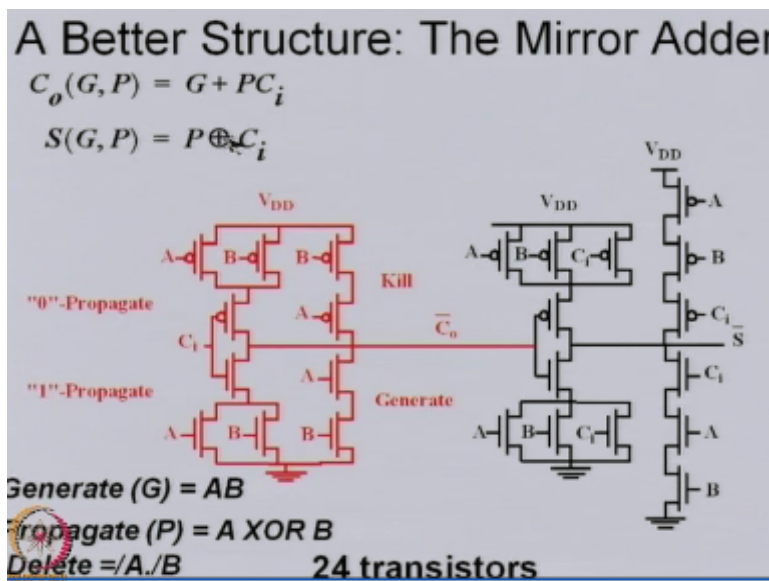
(Refer Slide Time: 52:46)



Ok. So, if you use blocks these are called odd cells, these are called even cells. So, you can see the first one instead of just C0 you put C0 Bar now this can have A bar B bar kind and then you can keep generating. So odd cells should receive bars and even cells should receive non bars. So, if you can do this, the delay on this will be lesser than this. And therefore, so you can have overall power reduction by using an inverter property.

Of course, we will come back tomorrow once again and we look into another possibility maybe we can talk this quickly.

(Refer Slide Time: 53:30)



This is called Mirror Adder. Actually, first time I am trying to implement the functions which are generated $G + PCI$ is the output carry and PC of CI is the sum function of G and P . And this particular input which is shown here. You have an CI input here which is in some kind of a series gate here. Your A and B which is shown here $A B$ shown here.

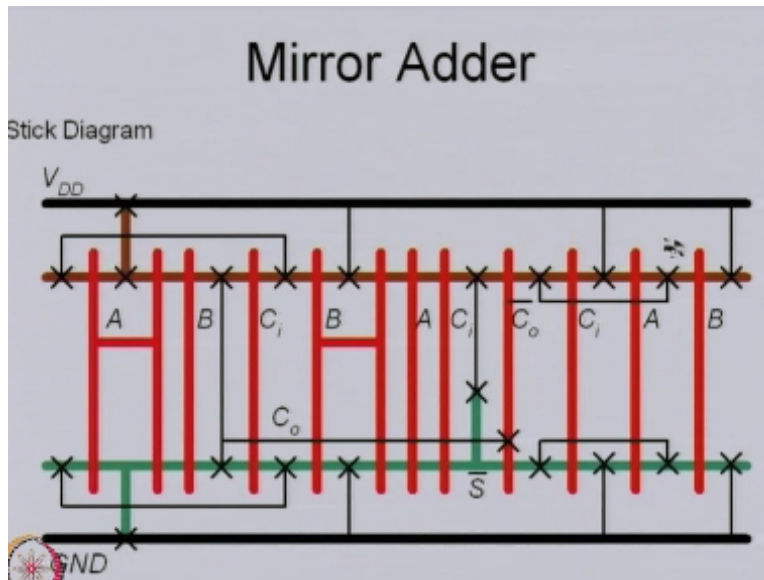
Please remember it is P channel N channels have same inputs whereas across this we have AB in series and AB in series and CI is your propagate. So, the idea is this if A is 0 and B 0 We want to go to 0 independent about this A 0 and B 0 this is going to be 1 and this is the bar that means 0 is propagating, irrespective what happens here if A and B are 0 . This N channel this P channel device conducts this node goes to VDD .

And if this goes to VDD the complement of CI which is Kill I want to make it 0 . So, 0 means bar of that is 0 . So, I actually killed this. You can see how I generate this. If A and B are 1 and then only I generate and if A and B are 1 this node goes to 0 because of N channel turns on so you generate. In case A is 0 and B is 1 , either of the CD chain is off and in that case depending on CI if A 0 this will be forming an inverter A 0 .

So, this will be blocked but since VDD will come here the output will be carried. Alternatively, it wanted 1 to appear in the then this will be on and this will be 0 and this output will be bar of that. So, I can propagate 0 I can propagate 1 I can kill or I can generate. Using this carry I can then put my standard logic once again similar ones with $ABCI$ and $ABCI$ with the new carry which is generated here, feed here and keep generating sums out of that.

This is only you can see from here, this is in serial order you can keep putting A and B in Blocks and at the end of the day you get $C0$ and S bar out coming from this. Total number of transistor in a mirror adder 24 compared to 28 earlier as shown.

(Refer Slide Time: 56:09)



And if you see a stick diagram it shows here, this is VDD line this is my ground line these are the polylines which form A this is my BCI and the same circuit we are using standard theorems we can always make a good looking layout of this just blow this stick diagram.

(Refer Slide Time: 56:29)

The Mirror Adder

- The NMOS and PMOS chains are *completely symmetrical*. A maximum of two series transistors can be observed in the carry-generation circuitry.
- When laying out the cell, the most critical issue is the minimization of the capacitance at node C_0 . The reduction of the diffusion capacitances is particularly important.
- The capacitance at node C_0 is composed of four diffusion capacitances, two internal gate capacitances, and six gate capacitances in the connecting adder cell.
- The transistors connected to C_1 are placed closest to the output.
- Only the transistors in the carry stage have to be optimized for optimal speed. All transistors in the sum stage can be minimal size.

Before we end this, we say what is the advantage of Mirror Adder. The NMOS and PMOS changed you can look at it everywhere whatever is here is opposite of here and whatever is here is exactly opposite of here in the sense opposite means P channel is replaced by N channel. This is symmetric and that is the biggest advantage in any implementation. Then, the NMOS and PMOS chains are completely symmetrical maximum two series transistors can be observed in the carry generation circuitry.

When laying out the cell, most critical issue is the minimization of the capacitance at node C0. Please remember this node is the most crucial which has to have the faster time because this carry is the major time. so how fast this capacitance is coming from here, coming from here coming from here and coming from here, so what is the node capacitor and the wire of course at this, will decide the propagation time in a great way.

While laying out the most critical issue is the minimization capacity node. See the reduction on diffusion capacity is very important, particularly very important the capacitance the C0 is already set compact composed I already said diffusion capacitance to internal gate capacitance 6 gate capacitance in connecting the adder cell.

The transistor connected to CI. Our place closest to the output this has to be understood wherever CI this should be always closest to output we know very well in our design whichever because this line is the evaluation stage, so this should come the last, so it should pre charge before next one starts similarly this should pre charge otherwise this will take time to fully charge or discharge the parts. So always in normal logic implementation the last one which appears is the closest at the output which is standard technique which we use here.

Only the transistor in carry stage has to be optimized. This S is not required because S required is not very much. The minimum stage all transistors are small the minimum size the only thing you require is the larger size transistor for delay stage optimization. So, we will come back a little later over the others of adders which we can implement and actually see to it that they may better or worse than the different kinds of adder architectures which will discuss later.

Thanks for the day.