KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

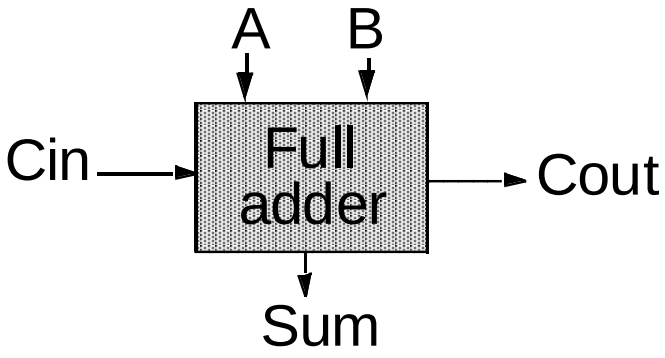
Website: www.ktunotes.in

4th Module

ADARSH K S

Adders

Full-Adder (FA)



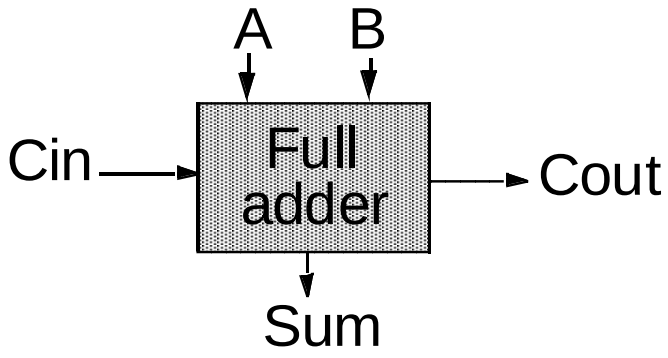
A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

Generate (G) = AB

Propagate (P) = A⊕ B

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

The Binary Adder



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

Generate (G) = AB

Propagate (P) = A $\oplus$ B

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

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

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

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

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

$$C_O = AB + BC_i + C_iA$$

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

$$C_O = AB + C_i(A+B)$$

$$S = ABC_i + \overline{C}_O(A+B+C_i)$$

CMOS Implementation

The Full Adder :-Circuit Design consideration

Static Adder circuit

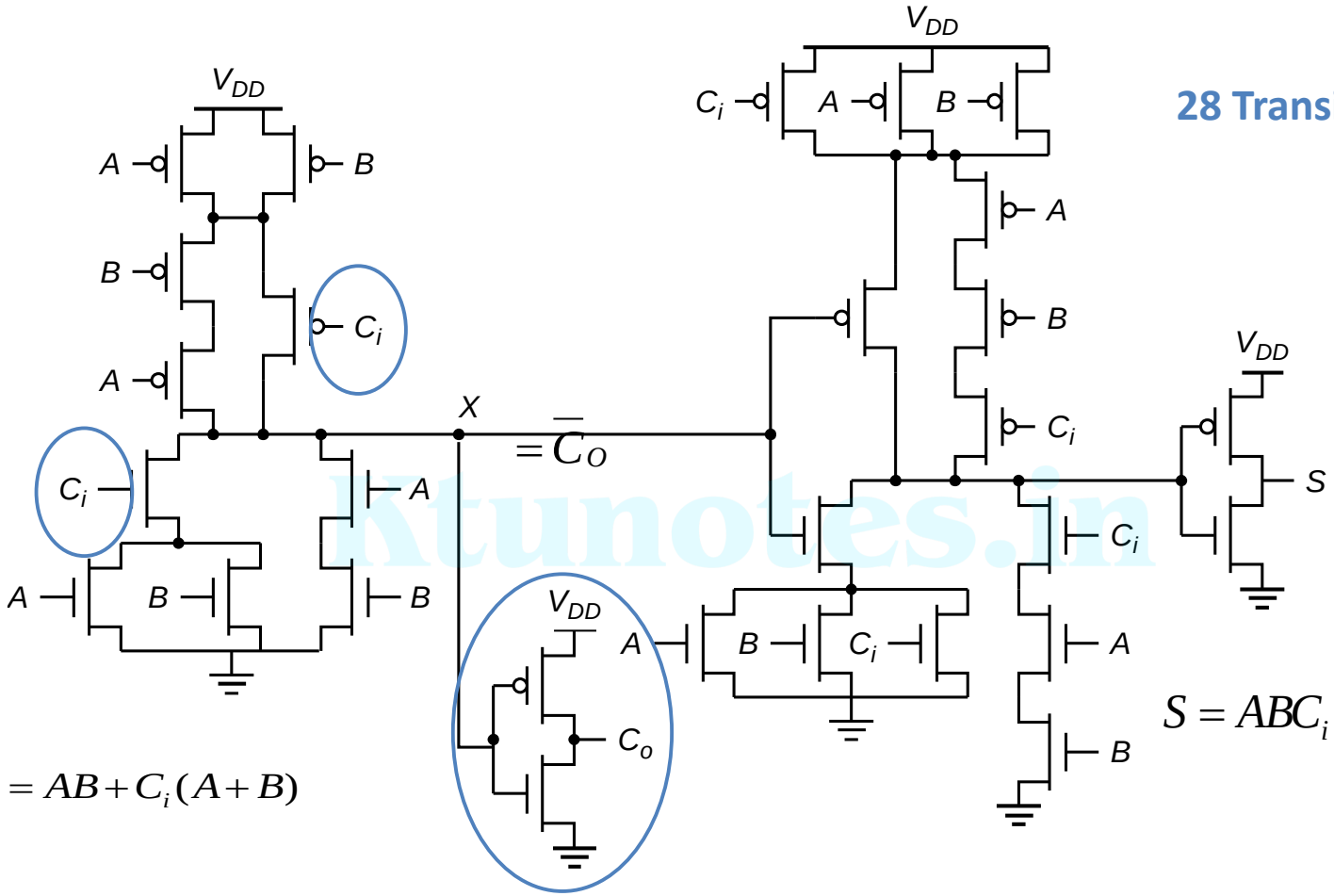
- One way to implement full adder circuit is take the logic equation and translate directly in to complementary CMOS circuitry
- Adjust to reduce transistor count
- It is advantageous to share some logic between the sum and carry generation sub circuits, as long as it does not slow down the carry generation
- Eg for such reorganized equation

$$C_O = AB + C_i(A + B)$$

$$S = ABC_i + \overline{C}_O(A + B + C_i)$$

- Corresponding adder design using complementary static cmos is shown in fig

Complimentary Static CMOS Full Adder



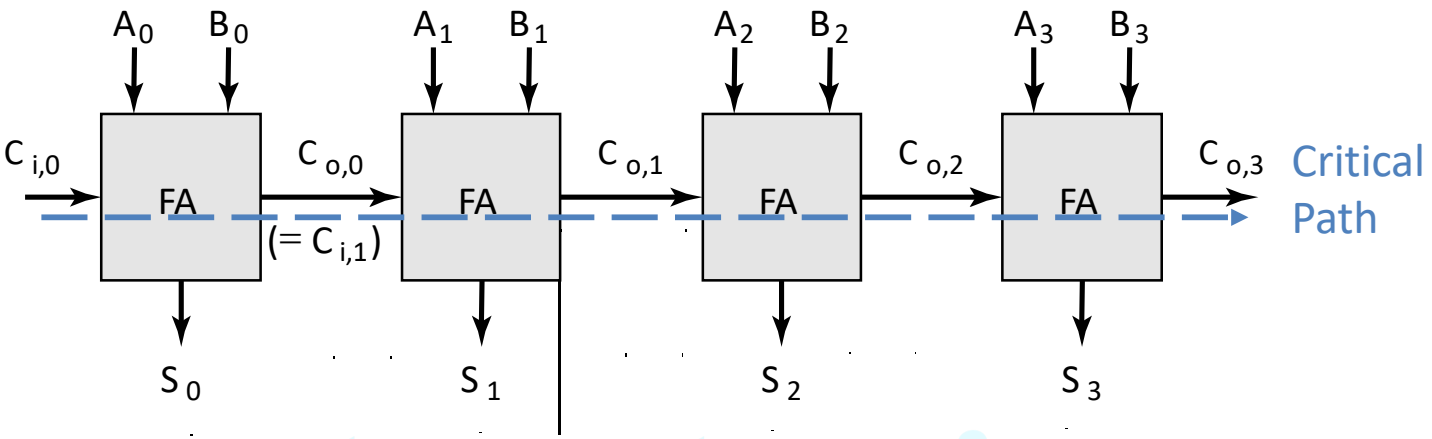
28 Transistors

$$C_O = AB + C_i(A+B)$$

$$S = ABC_i + \overline{C}_O(A+B+C_i)$$

- Signal propagate through two inverting stages in the carry generation circuit
- Ci is late arrival signal → near the output signal
- Co needs to be inverted → Slow down the ripple propagate

The Ripple-Carry Adder



Worst case delay linear with the number of bits

$$t_d = O(N)$$

$$t_{adder} = (N\text{-}1)t_{carry} + t_{sum}$$

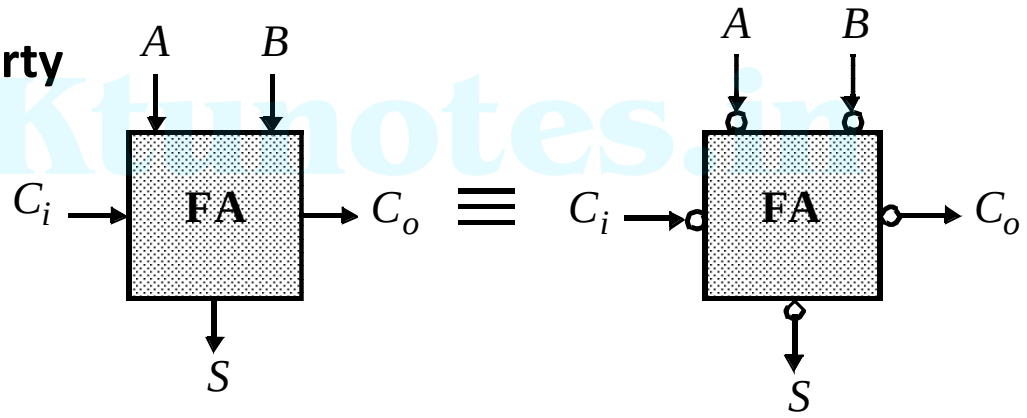
- **t**carry and **t**sum equal the propagation delay from Ci to Co and S, respectively
- Important to optimize tcarry than t sum

Goal: Make the fastest possible carry path circuit

- Nmos and pmos transistor are connected to ci are placed as close as possible to the output of the gate.
- This is a direct application of a circuit optimization technique
- The speed of this circuit can now be improved gradually by using some of adder properties

1. No of inverting stages in the carry path can be reduced by exploiting the inverting property(inverting all the inputs of full adder cell also inverts all the outputs)

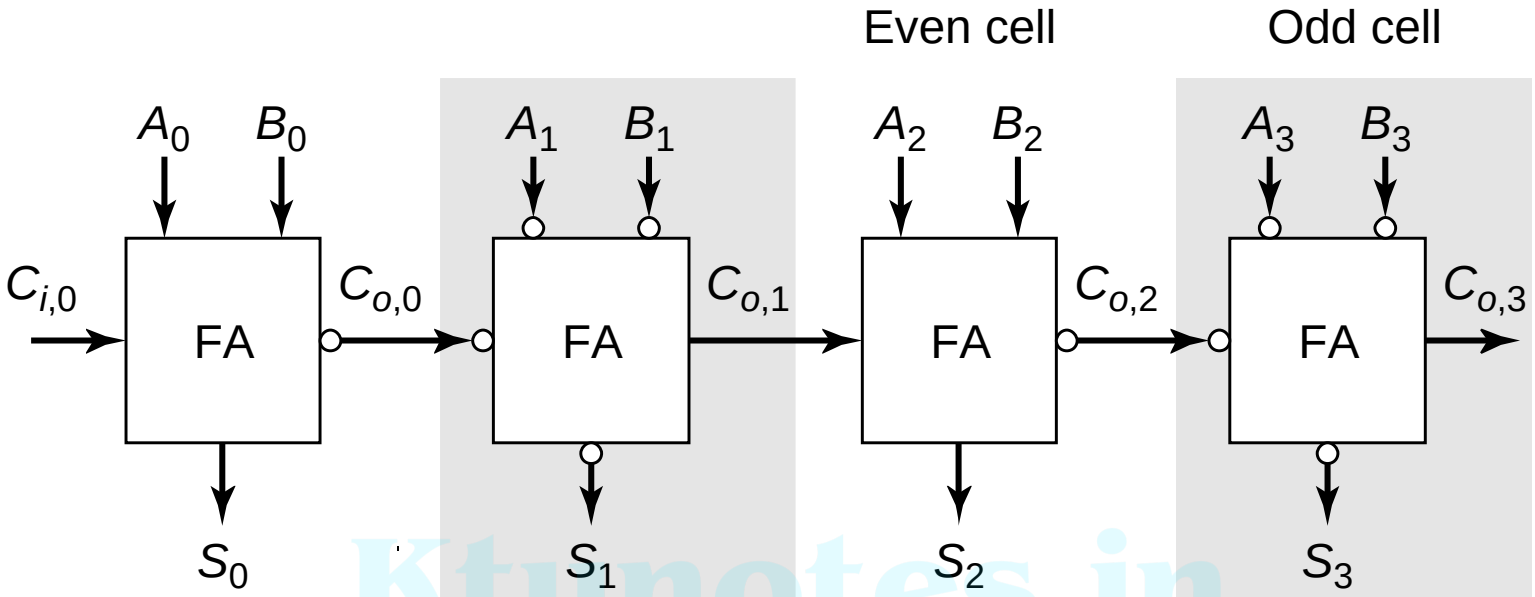
- **Inversion Property**



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

$$\overline{C_o}(A,B,C_i) = C_o(\bar{A},\bar{B},\bar{C_i})$$

Minimize Critical Path by Reducing Inverting Stages



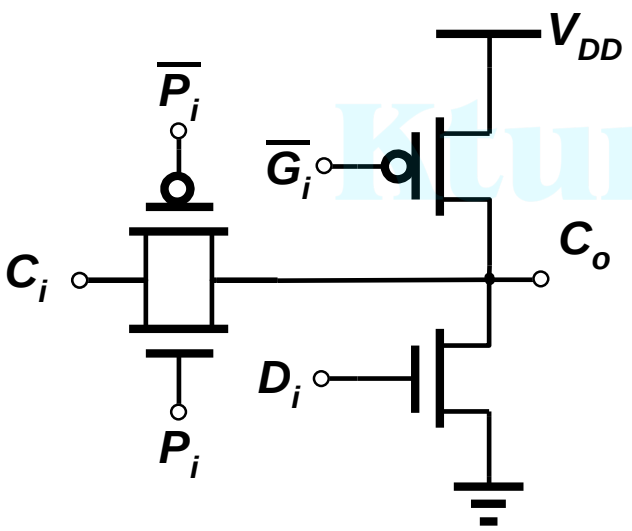
- **Exploit Inversion Property**
- **Reduce One inverter delay in each Full-adder (FA) unit**

A Better Structure:

1. The Mirror Adder design-24 transistors
2. Transmission-Gate Full Adder-24 transistors
3. Manchester Carry-Chain Adder

Manchester Carry-Chain Adder

Carry propagation circuitry can be simplified by adding generate and delete signals



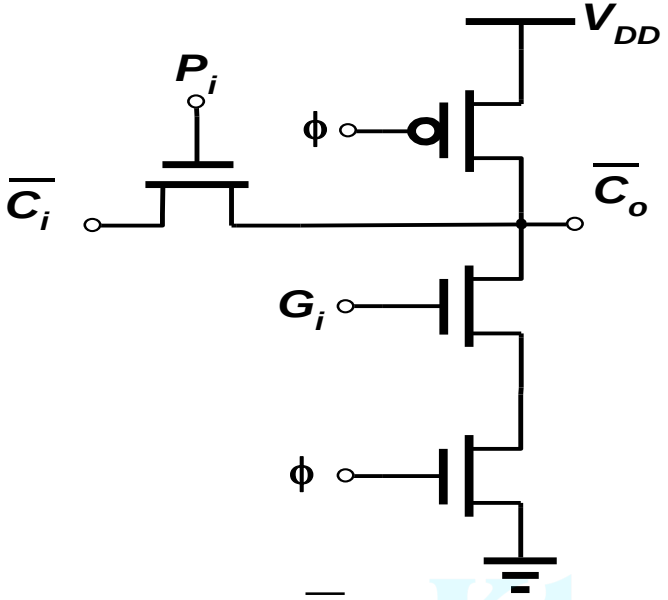
Static Circuits

Static stage

•This requires P to be generated as A⊕B(if propagate is true)
•If propagate condition not satisfied ,the output is either pulled low by Di or pulled up by Gi'

The Manchester adder stage improves on the carry-lookahead implementation.

$$C_O = (P \cdot C_i + G_i) \cdot \overline{D_i}$$



$$\begin{cases} \phi = 0, & \overline{C}_O = 1 \\ \phi = 1, & \overline{C}_O = P_i \overline{C}_i + \overline{G}_i \end{cases}$$

Dynamic stage

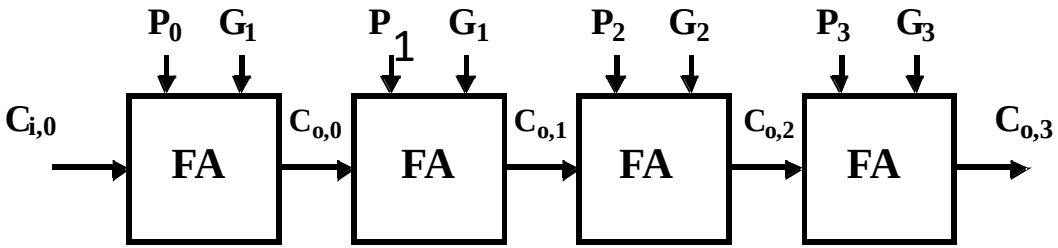
- Transmission gate can be replaced by NMOS only pass transistor
- When CLK is low, the output node is pre-charged by the p pull-up transistor.
- When CLK goes high, the pull-down transistor turns on.

If carry generate G=AB is true $\rightarrow$ the output node discharges.

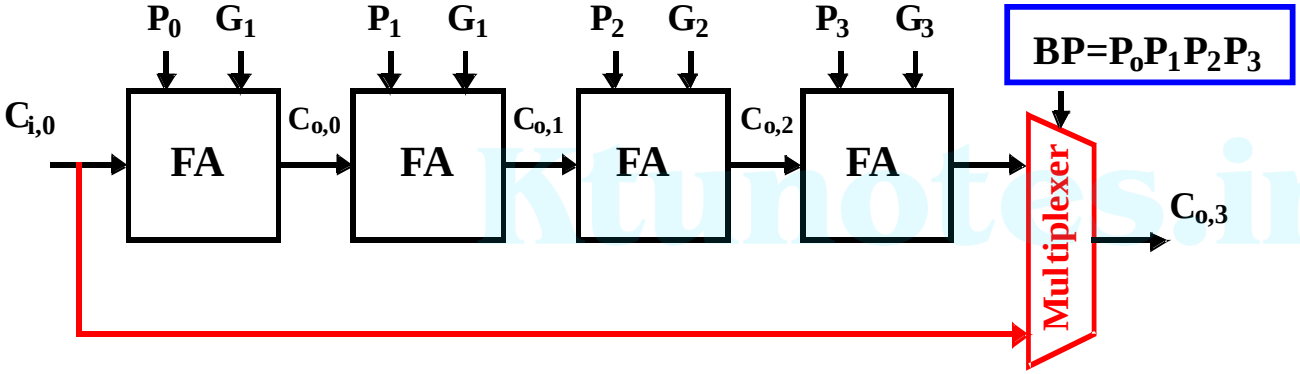
If carry propagate P=A+B is true $\rightarrow$ a previous carry may be coupled to the output node, conditionally discharging it.

- Ripple carry adder only for addition of small word length
- The linear dependence of adder speed on the number of bits makes the usage of ripple adder rather impractical
- Logic optimization is necessary

The Carry –bypass adder (Carry-Skip Adder)



Also called Carry-Skip



In a four bit adder block, Suppose the value of Ak and Bk(k=0.....3) are such that all propagate signals Pk(k=0..3) are high ,the incoming carry Ci,0=1 propagates under these condition through this adder chain and causes an outgoing carry Co,3=1

If ($P_0 \cdot P_1 \cdot P_2 \cdot P_3 = 1$) then Co,3=Ci,0
else Kill or Generate

Carry bypass in manchester carry chain adder

- The carry propagate either through the bypass path or a carry is generated in some where in the chain
- In both the cases delay is smaller than the normal ripple configuration

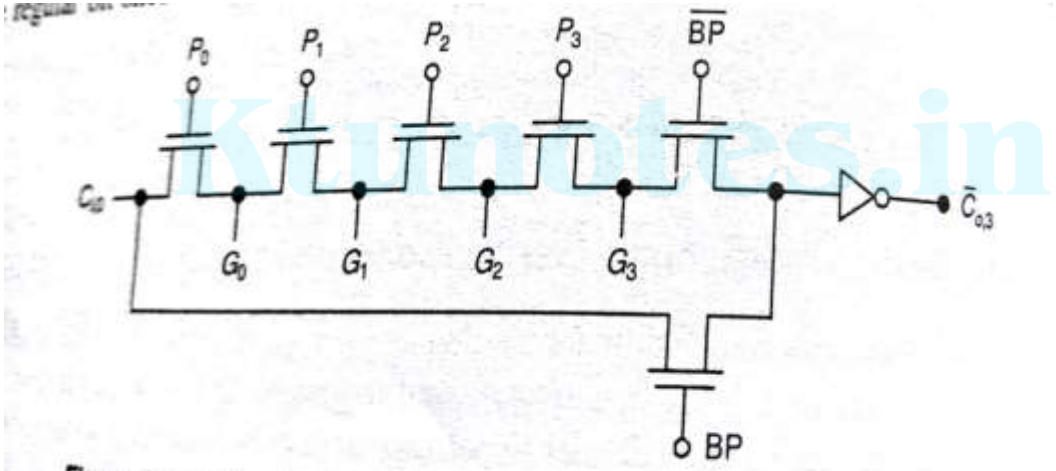
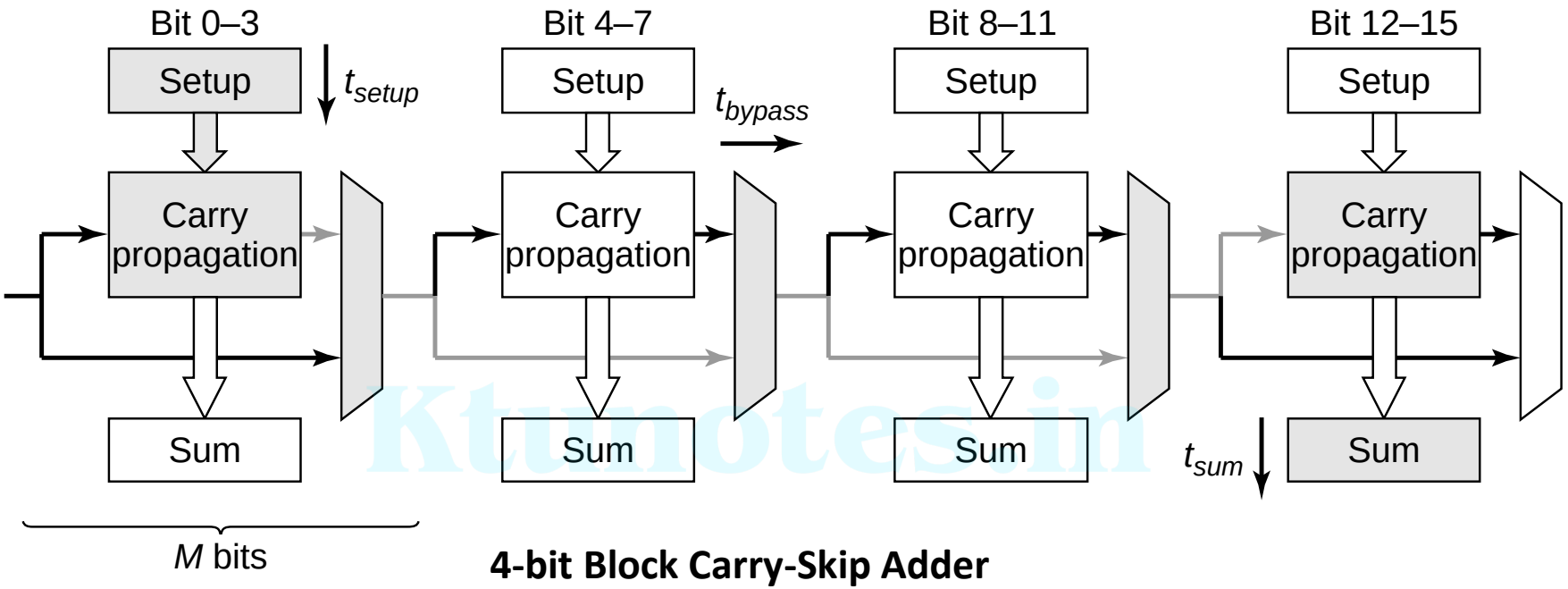


Figure 10.12 Manchester carry-chain implementation of bypass adder.

- For computing delay of an N-bit adder
- First assume the total adder is divided in to (N/M)equal-length bypass stages,each of which contains M bits



4-bit Block Carry-Skip Adder

- Worst-case delay → carry from bit 0 to bit 15 = carry generated in bit 0, ripples through bits 1, 2, and 3, skips the middle two groups ((N/M-2)), ripples in the last group from bit 12 to bit 15

- Total Propagation time can be derived from the above figure and is given by

$$td = t_{setup} + Mt_{carry} + (N/M-1)t_{bypass} + (M-1)t_{carry} + t_{sum}$$

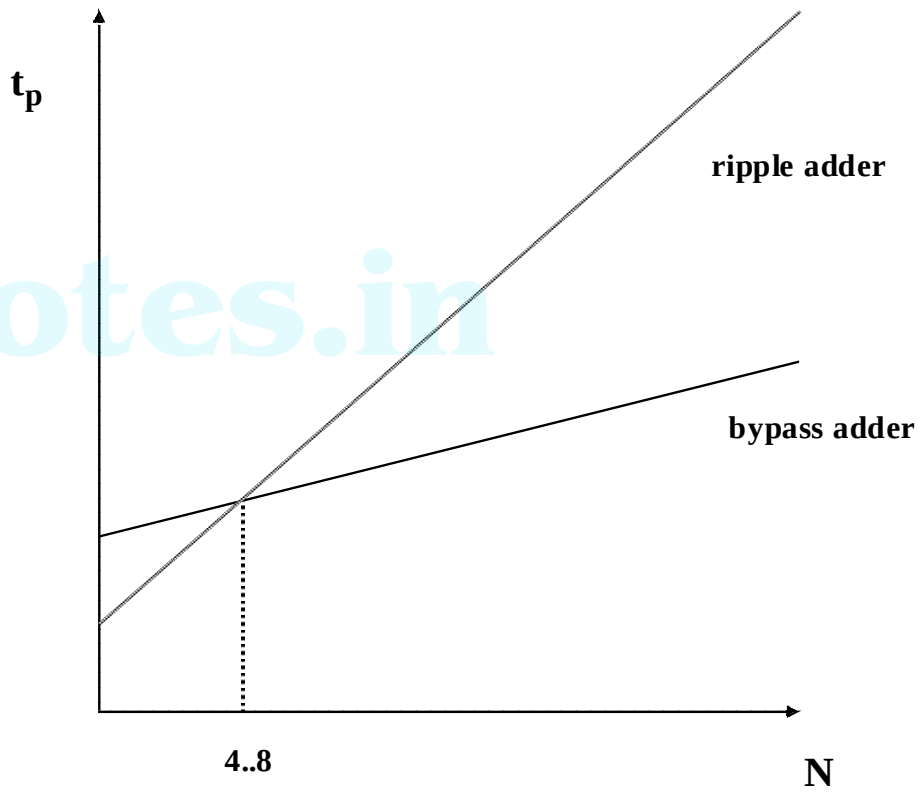
- t_{setup} : the fixed overhead time to create the generate and propagate signals
- t_{carry}: the propagation delay through a single bit
- t_{bypass}: the propagation delay through the bypass multiplexer of a single stage
- t_{sum} : the time to generate the sum of the final stage

tp is linear in the number of bits N
M bits form a Section → (N/M) Bypass Stages
The optimal no of bits per skip block is determined by technological parameter such as the extra delay of bypass selecting multiplexer ,the buffering requirements in the carry chain and the ratio of delay through the ripple and the bypass path

Carry Ripple versus Carry Bypass

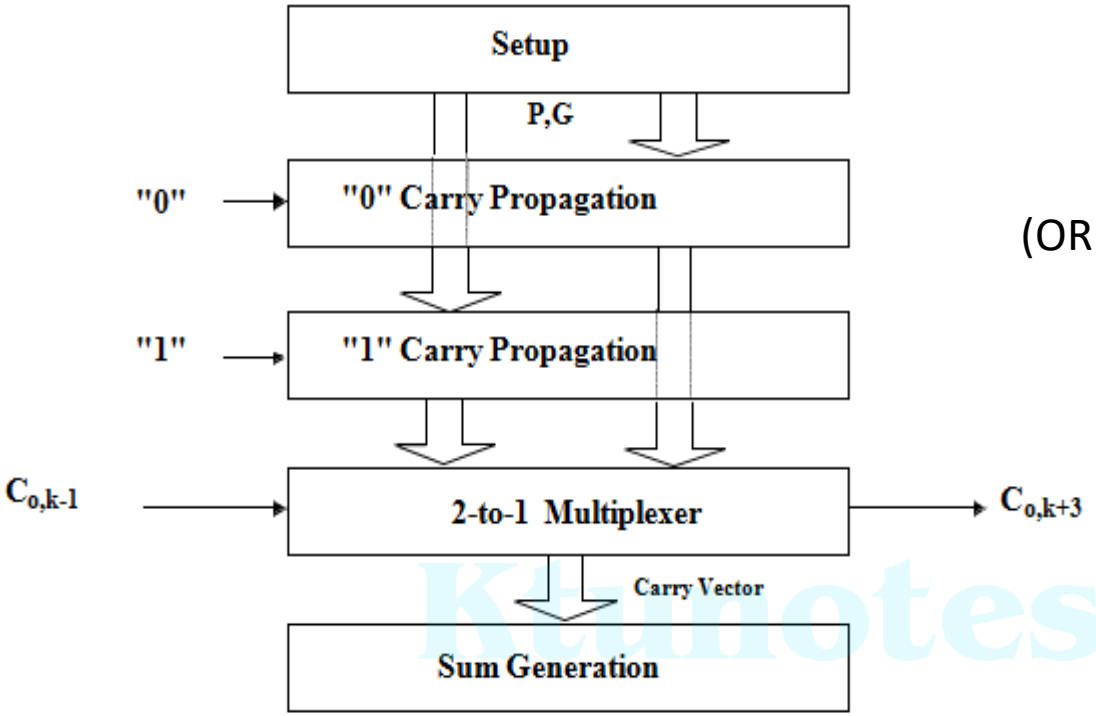
Wordlength (N) > 4~8 is better for Bypass Adder

The slope of delay function increases in a more gradual fashion than for ripple carry adder

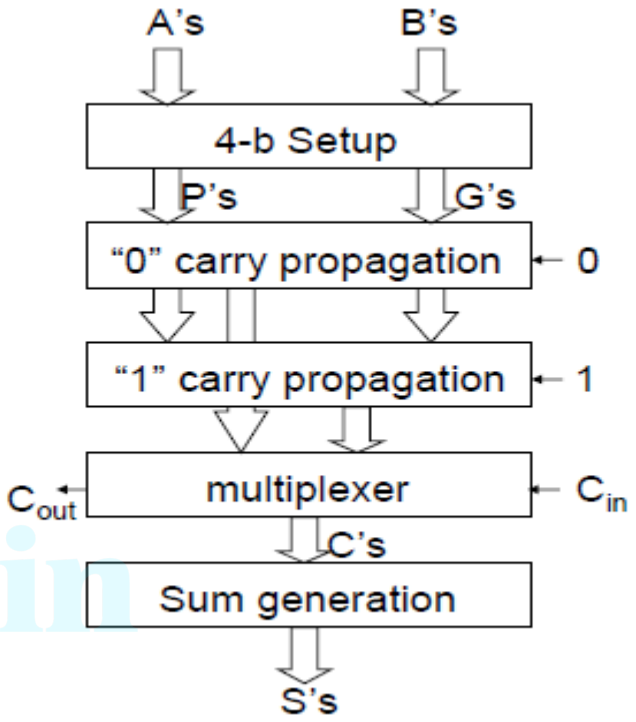


The Linear Carry Select adder

- In ripple carry –full adder cell has to wait for incoming carry before outgoing carry can be generated
- Solution to reduce this delay by evaluating both possible value of carry input in advance
- Once the real value of the incoming carry is known ,correct result is easily selected with a simple multiplexer stage
- Implementation of this idea appropriately called the carry select adder
- Consider a block of adder, which is adding bits k to k+3
- Insted of waiting for the arrival of the output carry of bit k-1,,both 0 and 1 possibilities are analysed
- Two carry paths are implemented
- When Co,k-1 finally settles, either the result of the 0 or the 1 path is selected by the multiplexer, which is performed with minimal delay



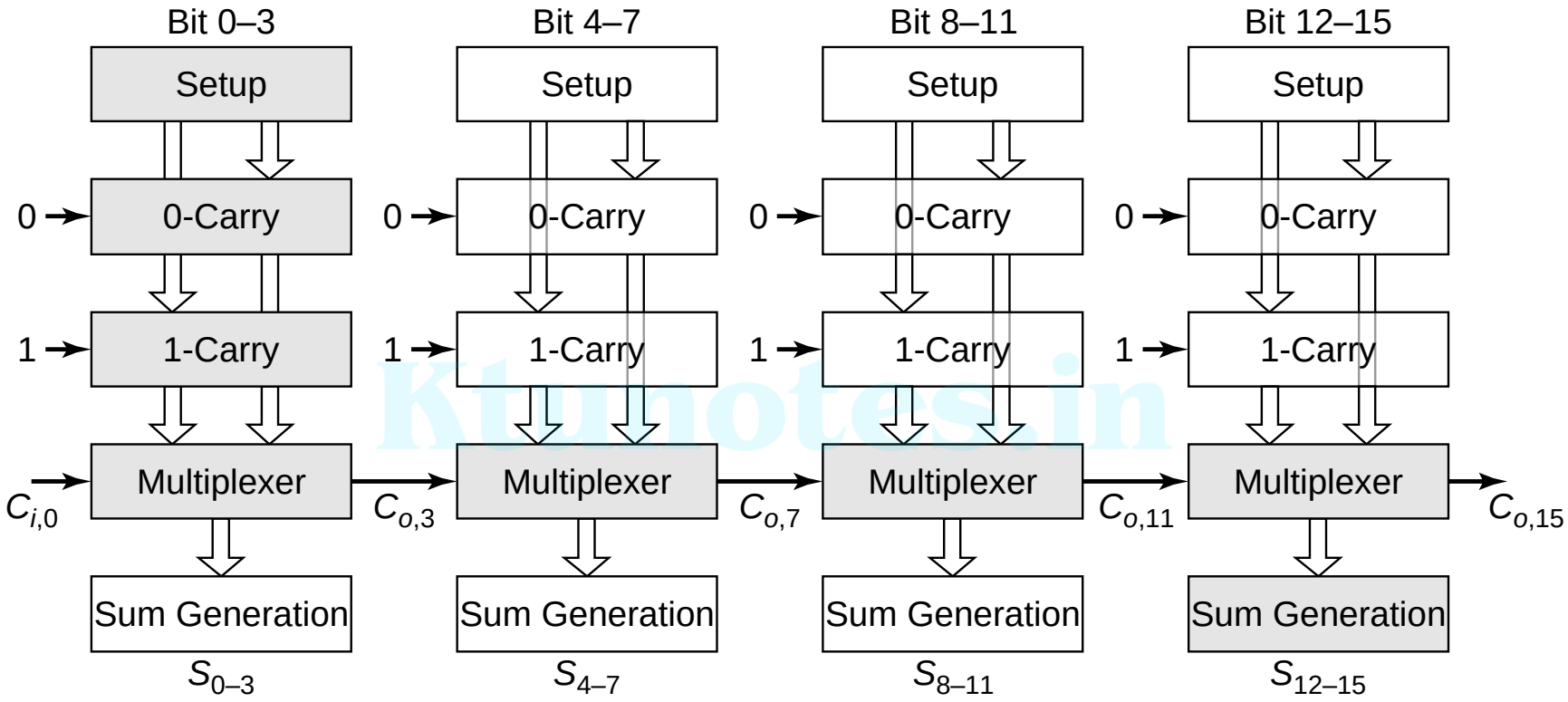
(OR)



•Hardware overhead of carry select adder is restricted to an additional carry path and a multiplexer, and equals 30% with respect to a ripple carry structure

•A full carry select adder is constructed by chaining a number of equal length adder stages ,as in carry bypass approach

•Critical path is shaded in gray



•From inspection of circuit ,we can derive a first order model of wrost case propagation delay of the module, written as

$$t_{add} = t_{setup} + \left(\frac{N}{M}\right) t_{carry} + M t_{mux} + t_{sum}$$

tsetup,tsum and tmux are fixed delays

N –total no .of bits

M-total no.of bits per stage

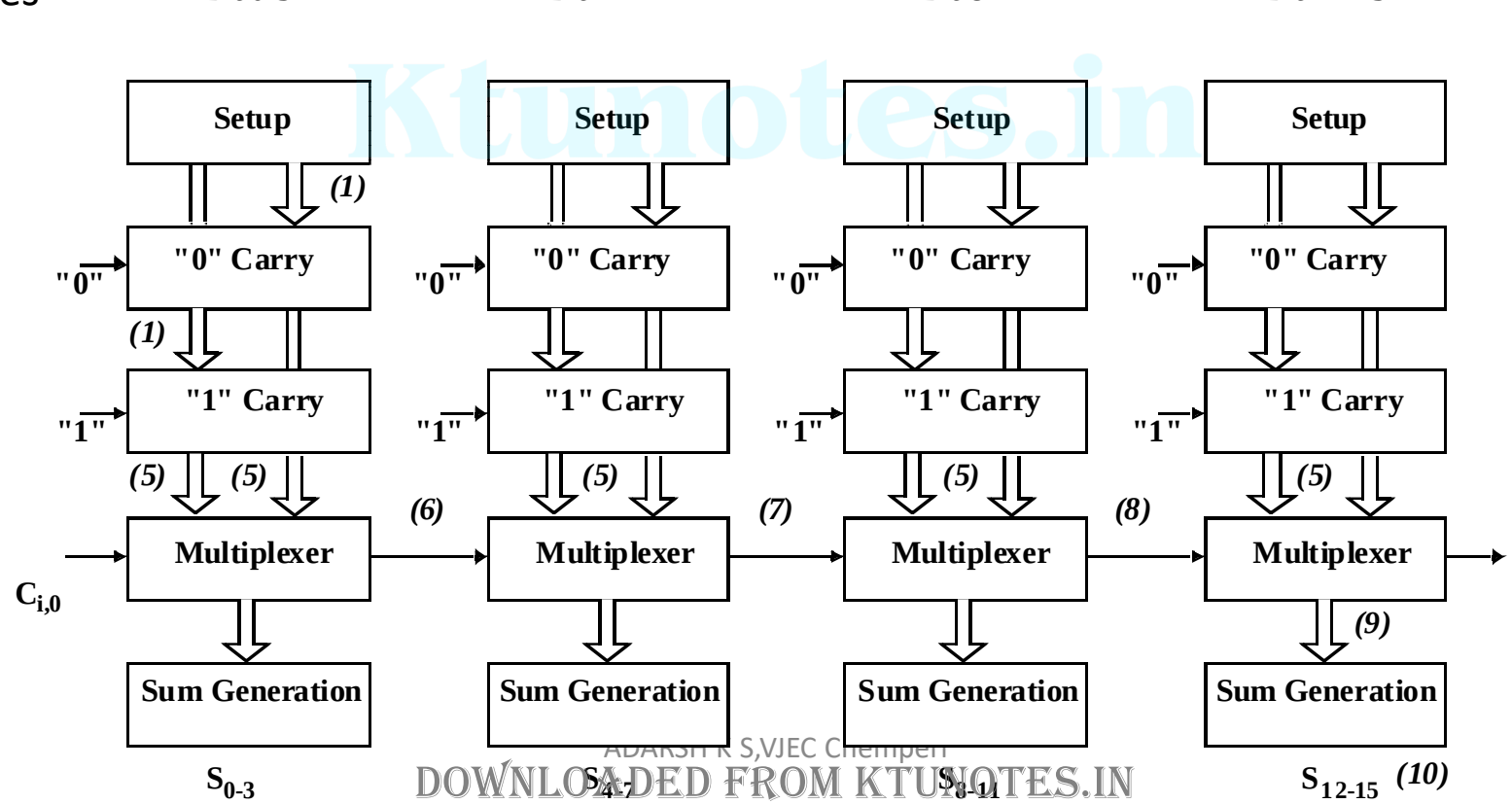
tcarry –delay of the carry through a single full adder cell

•Carry delay through a single block is proportional to length of that stage or equals Mtcarry

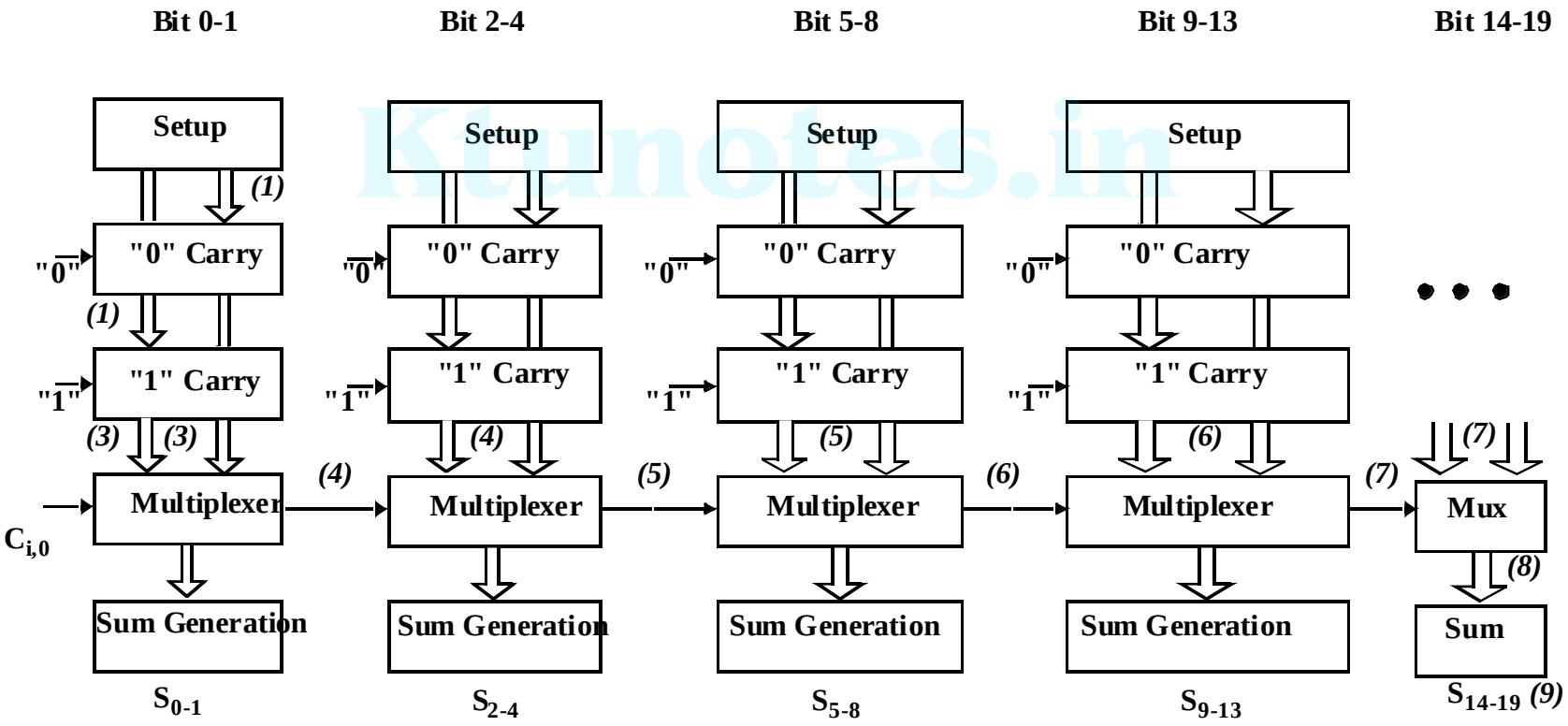
•The propogation delay of the adder is linearly proportional to N

Square Root Carry Select

- How an alert designer can major impact
- To optimise the design ,it is essential to locate the critical timing path first
- consider the case of a 16-bit linear carry select adder
- Assume that the full adder and multiplexer cells have identical propagation delay equals to a normalized value of 1
- The wrost case arrival times of the signals at the different network nodes with respect to the time the input is applied are marked
- i.e the critical path of adder ripples through the multiplexer network of the subsequent stages



•Major mismatch between the arrival time of the signals can be observed
•The result of the carry chains are stable, long before the multiplexer signal can be observed
•It make sense equalize the delay through both paths
•This can be achieved by progressively adding more bits to the subsequent stages in the adder ,requiring more time for the generation the carry signals
•For eg. first stage can add 2 bits, second contains 3,thrid has 4 and so forth as in fig



- The annotated arrival time shows that this adder topology is faster than the linear organization ,even though an extra stage is needed
- The same propagation delay is also valid for a 20-bit adder
- Discrepancy in arrival times at the multiplexer nodes can be eliminated
- Making adder stages progressively longer results in an adder structure with sub linear delay characteristics
- Assume N-bit adder with P stages: 1st stage adds M bits, 2nd has (M+1) bits

N= M+(M+1) + (M+2) + (M+3) +..... + (M+(P-1))

$$N = \frac{P^2}{2} + P(M - \frac{1}{2})$$

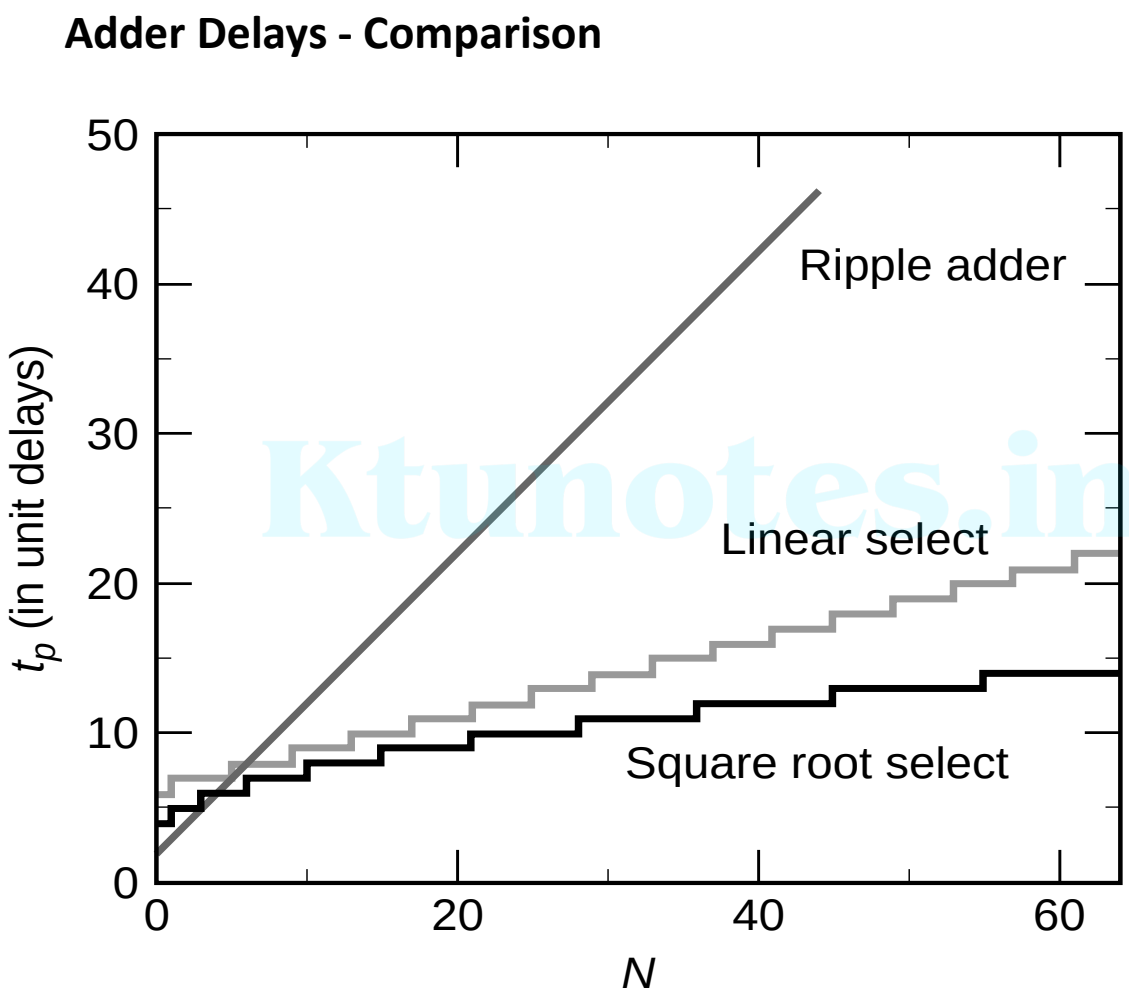
If M <<N , the first term dominate

$$\Rightarrow \quad P \approx \sqrt{2N}$$

Their for P can be used to express tadd as a function of N by rewriting tadd for carry select

$$t_{add} = t_{setup} + P \cdot t_{carry} + (\sqrt{2N}) t_{mux} + t_{sum}$$

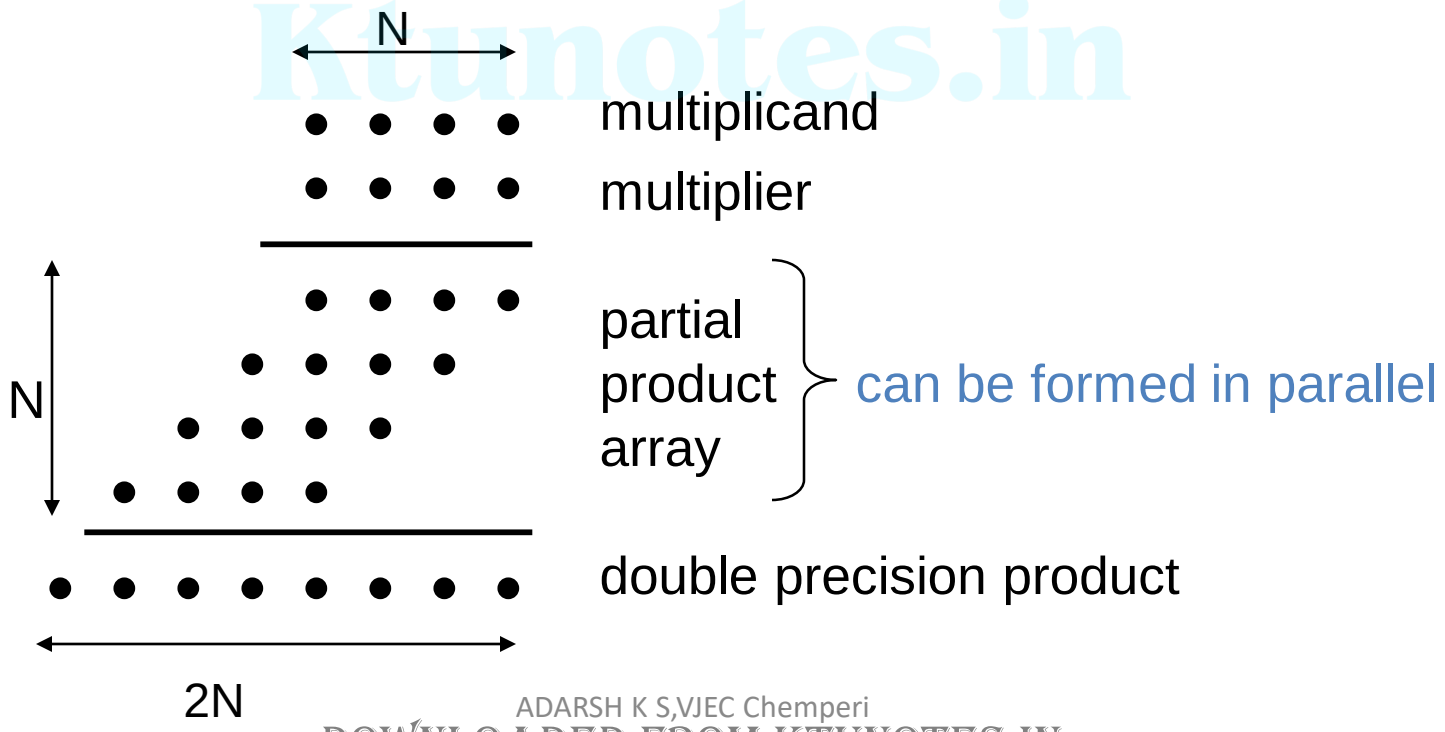
Delay is proportional to $\sqrt{N}$ for large adders



For large value of N, tadd become almost a constant for sruare root carry select adder

Multiplier

- A binary multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers
- A variety of computer arithmetic techniques can be used to implement a digital multiplier
- Most techniques involve computing a set of partial products, and then summing the partial products together



- The first stage of most multipliers involves generating the partial products which is nothing but an array of AND gates
- An n-bit by n-nit multiplier requires n2 AND gates for partial product generation
- The partial products are then added to give the final results

General Form

- Multiplicand: $Y = (y_{M-1}, y_{M-2}, \ldots, y_1, y_0)$
- Multiplier: $X = (x_{N-1}, x_{N-2}, \ldots, x_1, x_0)$
- Product:

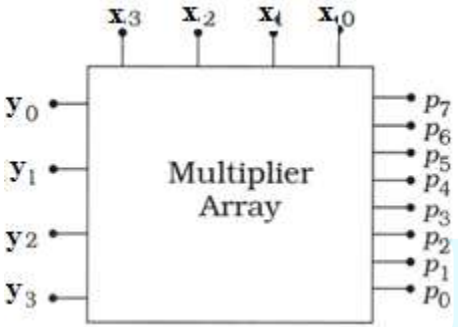
$$P = \left(\sum_{j=0}^{M-1} y_j 2^j \right) \left(\sum_{i=0}^{N-1} x_i 2^i \right) = \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} x_i y_j 2^{i+j}$$

						y_5	y_4	y_3	y_2	y_1	y_0	multiplicand
						x_5	x_4	x_3	x_2	x_1	x_0	multiplier
						x_0y_5	x_0y_4	x_0y_3	x_0y_2	x_0y_1	x_0y_0	
					x_1y_5	x_1y_4	x_1y_3	x_1y_2	x_1y_1	x_1y_0		partial products
				x_2y_5	x_2y_4	x_2y_3	x_2y_2	x_2y_1	x_2y_0			
			x_3y_5	x_3y_4	x_3y_3	x_3y_2	x_3y_1	x_3y_0				
		x_4y_5	x_4y_4	x_4y_3	x_4y_2	x_4y_1	x_4y_0					
	x_5y_5	x_5y_4	x_5y_3	x_5y_2	x_5y_1	x_5y_0						
p_{11}	p_{10}	p_9	p_8	p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product

- The simplest way to perform multiplication is to use a single two input adder
- For inputs that are M and N bits wide, the multiplication take M cycles, using an N bit adder
- This **shift and add algorithm** for multiplication add together M partial products
- Each partial product is generated by multiplying the multiplicand with a bit of multiplier---is an AND operation--- and shifting the result on the basis of the multiplier bit position
- All partial products are generated at same time and organized in an array
- A multi operand addition is applied to compute the final product
- The resulting structure is called an **array multiplier** combines of following three function: **partial product generation** , **partial product accumulation and final addition**

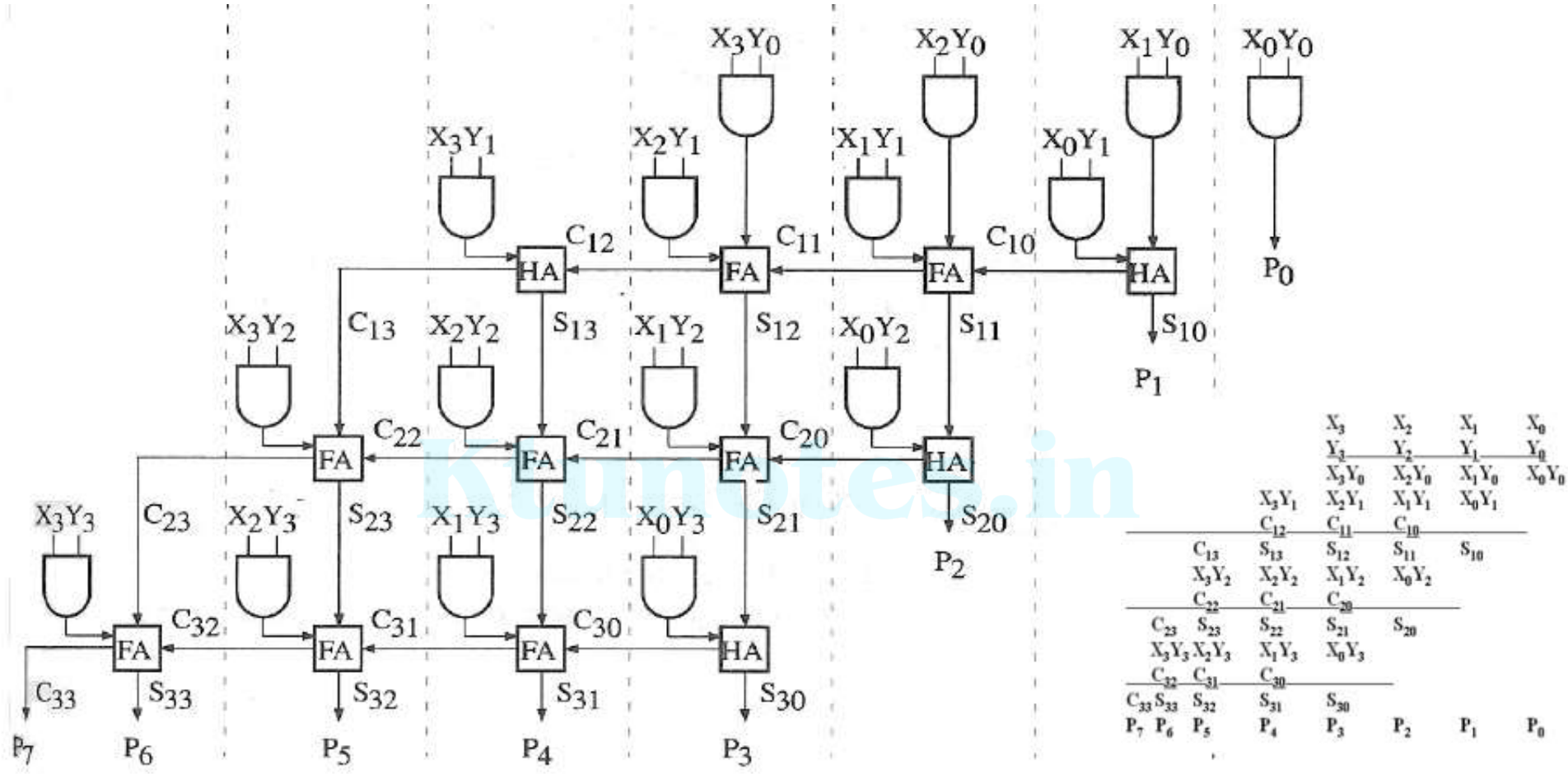
Array multiplier

- An array multiplier accepts the multiplier and multiplicant and uses an array of cells to calculate the bit product xi .yj individually in a parallel manner



			X_3	X_2	X_1	X_0		Multiplicand
			Y_3	Y_2	Y_1	Y_0		Multiplier
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0		Partial product 0
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1			Partial Product 1
		C_{12}	C_{11}	C_{10}				1st row carries
	C_{13}	S_{13}	S_{12}	S_{11}	S_{10}			1st row sums
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2				partial product 2
	C_{22}	C_{21}	C_{20}					2nd row carries
C_{23}	S_{23}	S_{22}	S_{21}	S_{20}				2nd row sums
X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3					partial product 3
C_{32}	C_{31}	C_{30}						3rd row carries
C_{33}	S_{33}	S_{32}	S_{31}	S_{30}				3rd row sums
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	Final Product

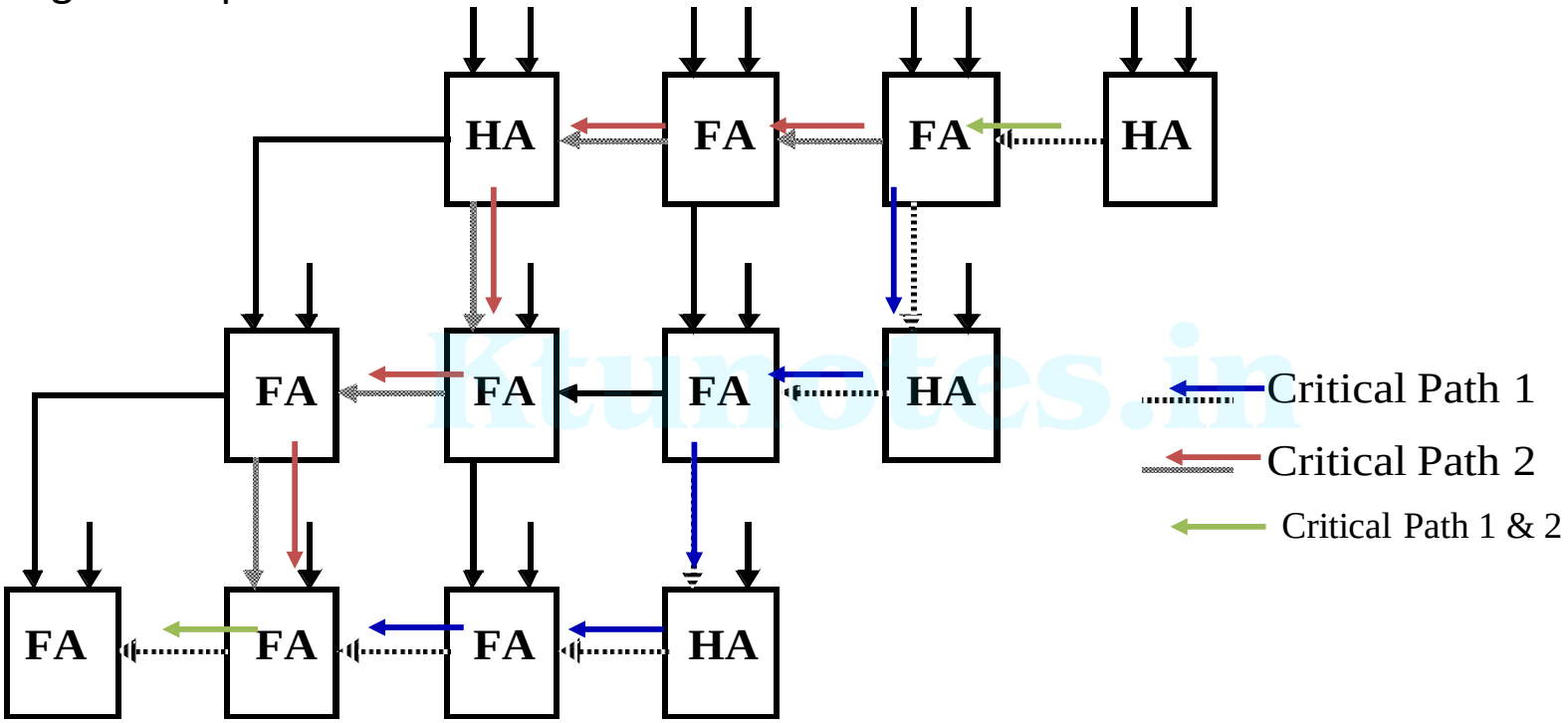
Block diagram of 4X4 Array Multiplier



- n bit x n bit multiplication would require n2 AND gates, n(n-2) full adders and n half-adders.
- due to the array organisation ,determination of delay of this circuit is not straight forward

Operation

- Here the partial sum adders are implemented as ripple carry structures
- Performance optimization requires that the critical timing path be identified first
- Large no of path of identical can be identified



- Here the propagation delay can be expressed as

$$t_{mult} \approx [(M-1)+(N-2)]t_{carry} + (N-1)t_{sum} + (N-1)t_{and}$$

tcarry is the propagation delay between input and output carry
tsum is the delay between the input carry and sum bit of full adder
tand is the delay of the AND gate

- Since all critical paths have the same length,speeding up just one of them- by replacing one adder by a faster one such as a carry select adder
- All critical path have to be attacked at the same time
- From the equation , minimization of **tmult** requires the minimization of both **tcarry** and **tsum**
- it could beneficial for tcarry to equal tsum