High Speed Adders,

1. Carry skip adders
2. Carry select adders
3. Carry save adders

Carry skip adders:

A carry-skip adder consists of a simple ripple carry-adder with a special speed up carry chain called a **skip chain**. This chain defines the distribution of ripple carry blocks, which compose the skip adder. The addition of two binary digits at stage i, where i is not equal to 0, of the ripple carry adder depends on the carry in, ***Ci*** , which in reality is the carry out, ***Ci-1***, of the previous stage. Therefore, in order to calculate the sum and the carry out, ***Ci+1*** , of stage i, it is imperative that the carry in, ***Ci***, be known in advance. It is interesting to note that in some cases ***Ci+1*** can be calculated without knowledge of ***Ci***.

Boolean Equations of a Full Adder:

Pi = Ai ⊥ Bi Equal. 1 --carry propagate of it stage

Si = Pi ⊥Ci Equal. 2 --sum of it stage

Ci+1 = AiBi + PiCi Equal. 3 --carry out of it stage

Supposing that ***Ai*** = ***Bi***, then ***Pi*** in equation 1 would become zero (equation 4). This would make ***Ci+1*** to depend only on the inputs ***Ai*** and ***Bi***, without needing to know the value of ***Ci***. **Ai = BI; Pi = 0** Equal. 4 --from #Equation 1

If **Ai = Bi = 0 ; Ci+1 = AiBi = 0** --from equation 3

If **Ai = Bi = 1; Ci+1 = AiBi = 1** --from equation 3

Therefore, if Equation 4 is true then the carry out, ***Ci+1***, will be one if ***Ai*** = ***Bi*** = ***1*** or zero if ***Ai***= ***Bi*** = ***0***. Hence the output can be computed with the carry out at any stage of the addition provided equation 4 holds. These would enable to build an adder whose average time of computation would be proportional to the longest chains of zeros and of different digits of A and B.

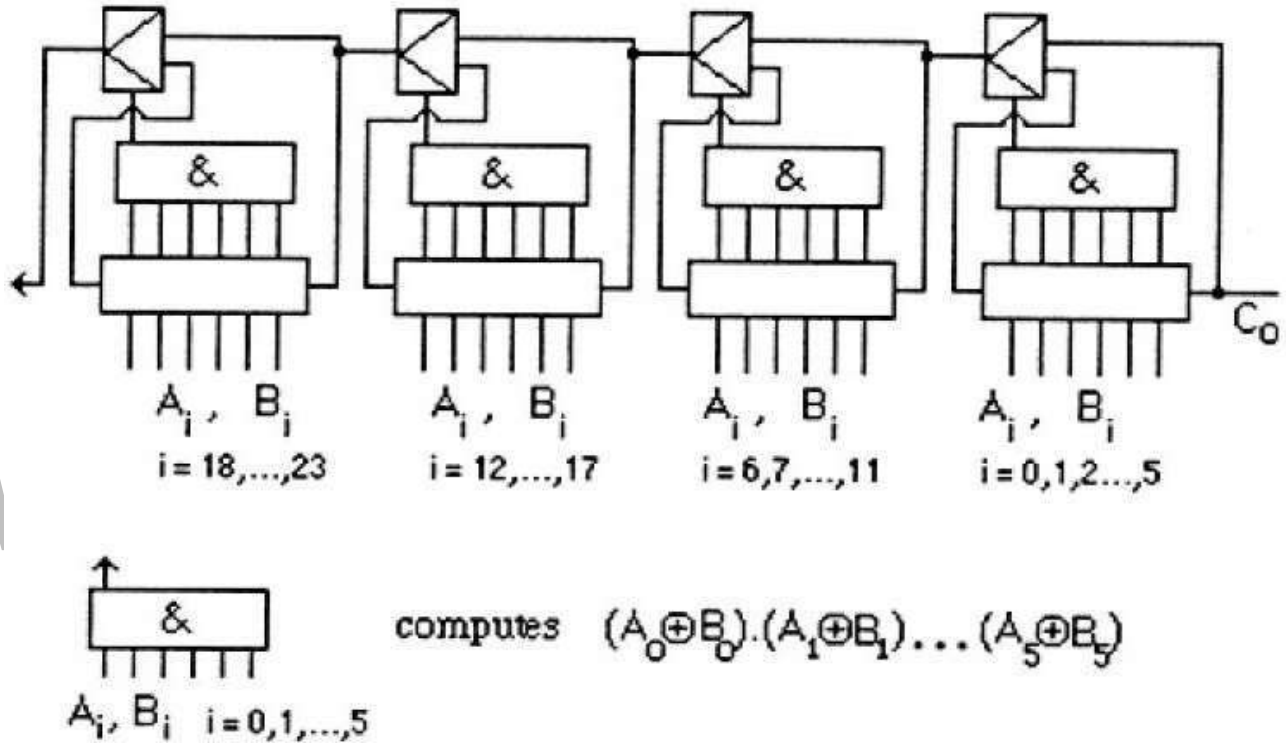


Fig4.1.1: Carry skip Chain

[Source: Jan M. Rabies ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits: A Design perspective]

Carry select adders:

The concept of the carry-select adder is to compute alternative results in parallel and subsequently selecting the correct result with single or multiple stage hierarchical techniques. In order to enhance its speed performance, the carry-

Select adder increases its area requirements. In carry-select adders both sum and carry bits are calculated for the two alternatives: input carry “0” and “1”. Once the carry-in is delivered, the correct computation is chosen (using a MUX) to Produce the desired output. Therefore instead of waiting for the carry-in to calculate the sum, the

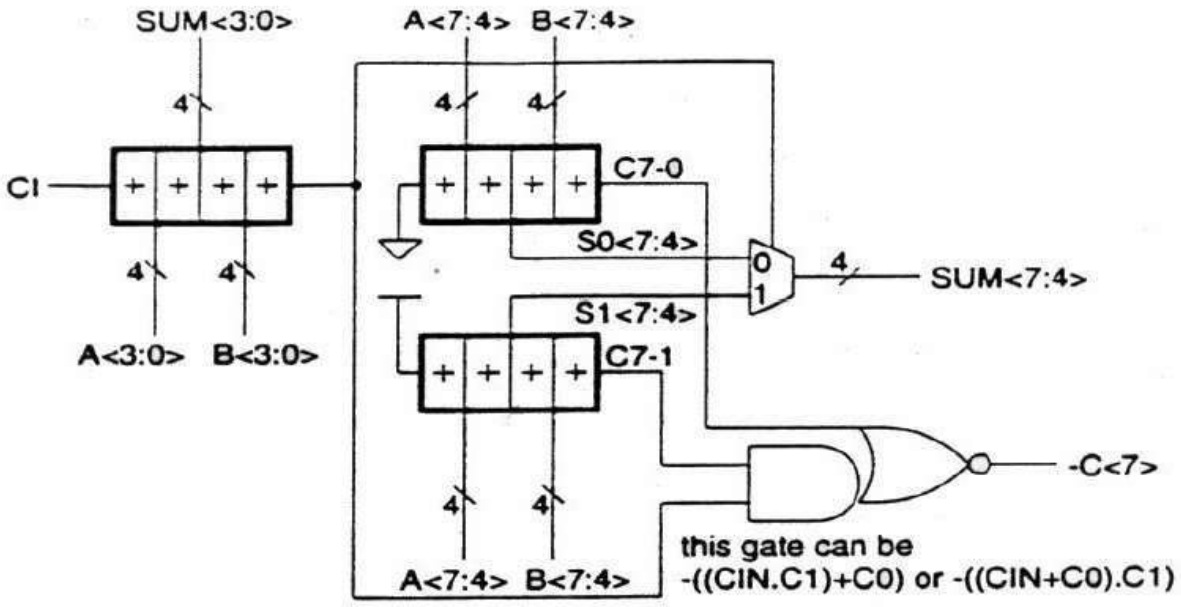


Fig 4.1.2: Concept of carry select adder

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

Sum is correctly output as soon as the carry-in gets there. The time taken to compute the sum is then avoided which results in a good improvement in speed. Carry-select adders can be divided into equal or unequal sections. For each section, the calculation of two sums is accomplished using two 4-bit ripple-carry adders. One of these adders is fed with a 0 as carry-in whereas the other is fed a

1. Then using a multiplexer, depending on the real carryout of the previous

Section, the correct sum is chosen. Similarly, the carryout of the section is computed twice and chosen depending of the carryout of the previous section. The concept can be expanded to any length for example a 16-bits carry-select adder can be composed of four sections. Each of these sections is composed of two 4-bits ripple-carry adders. This is referred as linear expansion.

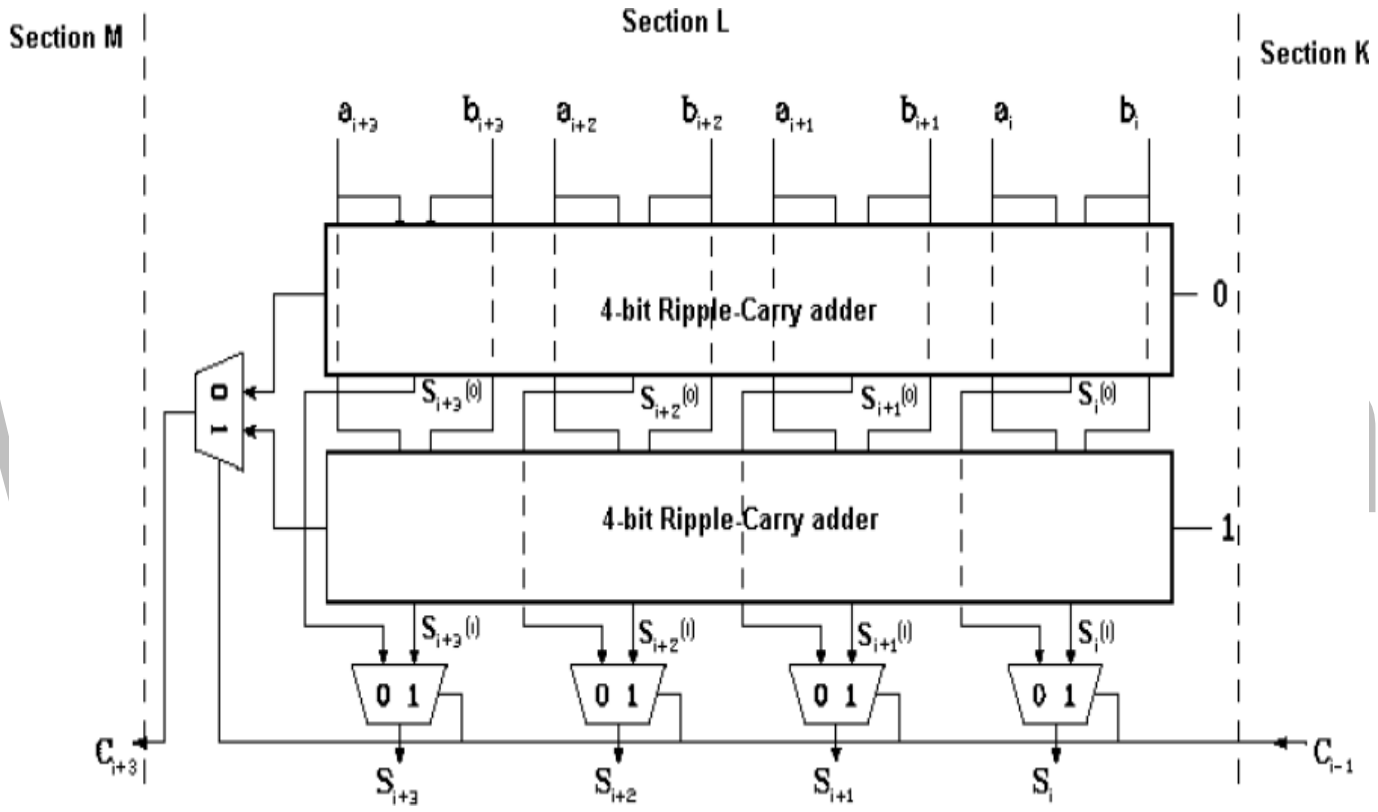


Fig 4.1.3: one section of a large carry select adder

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

Carry save adders:

In most computations, we need to add several operands together, carry save adders are ideal for this type of addition. A carry save adder consists of a ladder of standa lone full adders, and carries out a number of partial additions. The

Principal idea is that the carry has a higher power of 2 and thus is routed to the next column. Doing additions with Carry save adder saves time and logic.

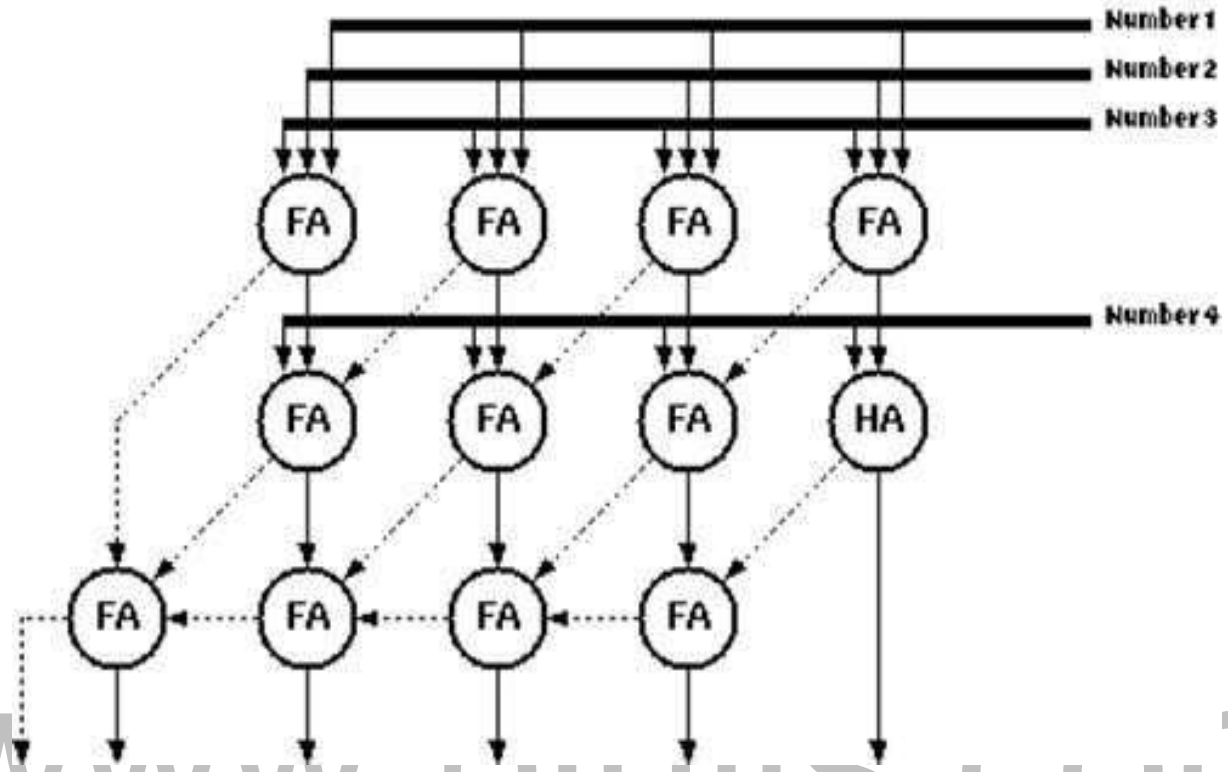


Fig 4.1.4: Carry save adder for 4 bit number

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

In this method, for the first 3 numbers a row of full adders are used. Then a row of full adders is added for each additional number. The final results, in the form of two numbers SUM and CARRY, are then summed up with a carry propagate adder or any other adder

Ripple Carry Adder

An N-bit adder can be constructed by casing N full adders as shown in

Fig.4.1.4 (a) for N=4. This is called a carry-ripple adder (or ripple-carry adder). The carry-out of bit i, Ci is the carry-in to bit i + 1. This carry is said to have twice the weight of the sum SI. The delay of the adder is set by the time for the carries to ripple through the N stages, so the $t_{C \rightarrow C_{OUT}}$ delay should be minimized.

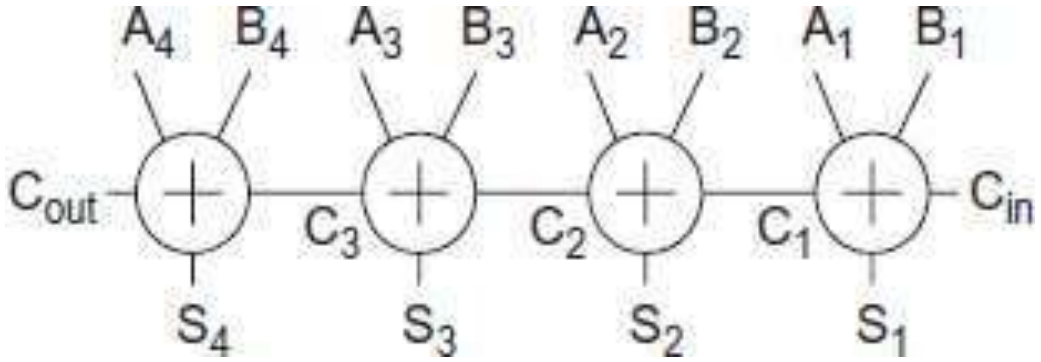


Fig.4.1.4 (a) 4-bit carry-ripple adder

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

In carry-ripple adders, the critical path goes from C to Cout through many full adders, so the extra delay computing S is unimportant. This delay can be reduced by omitting the inverters on the outputs. Fig.4.1 (b) shows the adder with transistor sizes optimized to favor the critical path using a numberof techniques:

- Feed the carry-in signal (C) to the inner inputs so the internal capacitance is already discharged.
- Make all transistors in the sum logic whose gate signals are connected to the carry-in and carry logic minimum size (1 unit, e.g., 4λ). This minimizes the branching effort on the critical path. Keep routing on this signal as short as possible to reduce interconnect capacitance.
- Determine widths of series transistors by logical effort and simulation. Build an asymmetric gate that reduces the logical effort from C to Cout at the expense of effort to S.

- Use relatively large transistors on the critical path so that stray wiring capacitance is a small fraction of the overall capacitance.
- Remove the output inverters and alternate positive and negative logic to reduce delay and transistor count to 24.

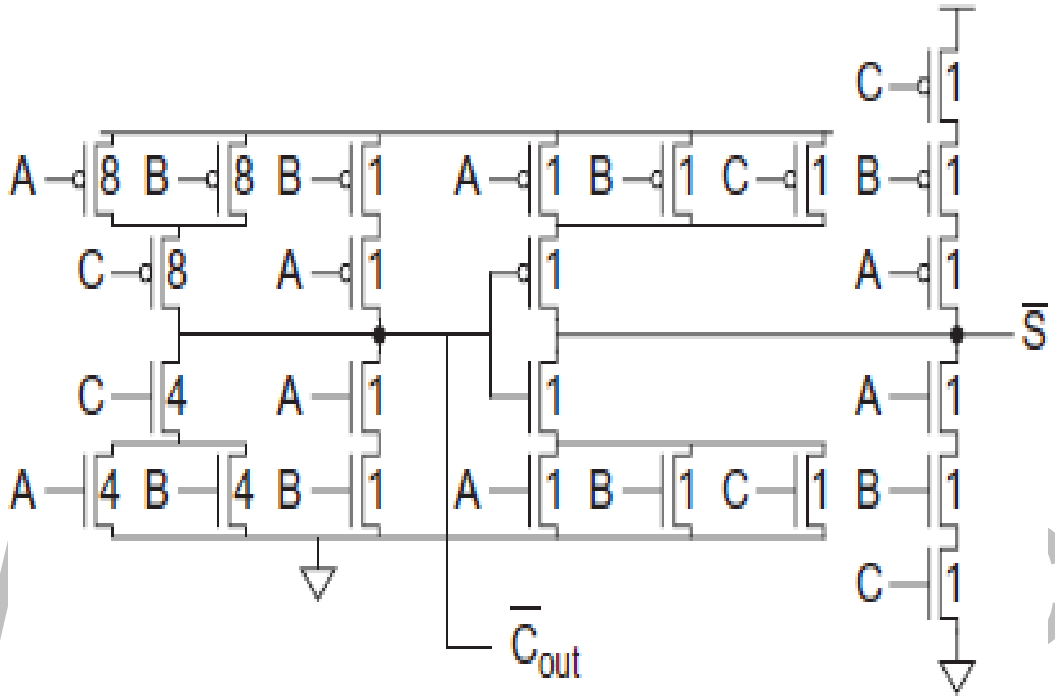


Fig.4.1.4 (b) Full adder for carry-ripple operation

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

This delay can be reduced by omitting the inverters on the outputs, as was done in Fig.4.1.4 (b). Because addition is a self-dual function (i.e., the function of complementary inputs is the complement of the function), an inverting full adder receiving complementary inputs produces true outputs. Fig.4.1.4 (c) shows a carry ripple adder built from inverting full adders. Every other stage operates on complementary data. The delay inverting the adder inputs or sum outputs is off the critical ripple-carry path.

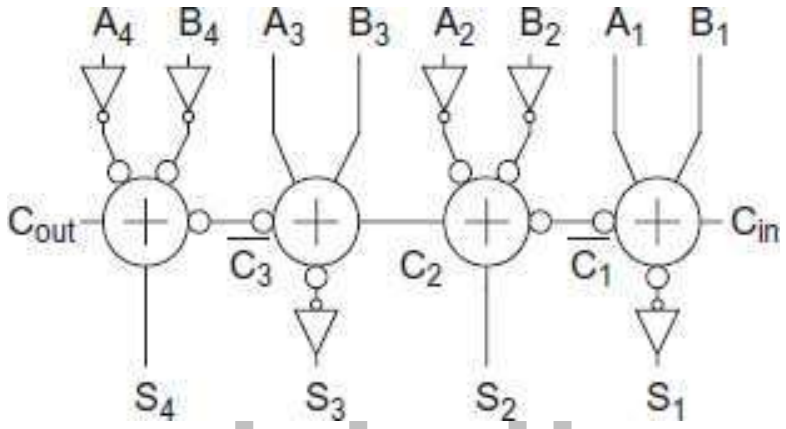


Fig.4.1.4 (c) 4-bit carry-ripple adder (inverting full adders)

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

- **RIPPLE CARRY ADDITION USING PG**

The critical path of the carry-ripple adder passes from carry-in to carry-out along the carry chain majority gates. As the P and G signals will have already stabilized by the time the carry arrives, we can use them to simplify the majority

$$C_i = A_i B_i + (A_i + B_i) C_{i-1}$$
$$= A_i B_i + (A_i \oplus B_i) C_{i-1}$$
$$= G_i + P_i C_{i-1}$$

Function into an AND-OR gate:

Because Ci= Gi: 0, carry-ripple addition can now be viewed as the extreme case

Of group

$$G_{i:0} = G_i + P_i \cdot G_{i-1:0}$$

PG logic in which a 1-bit group is combined with an i-bit group to form an (i+1) bit group

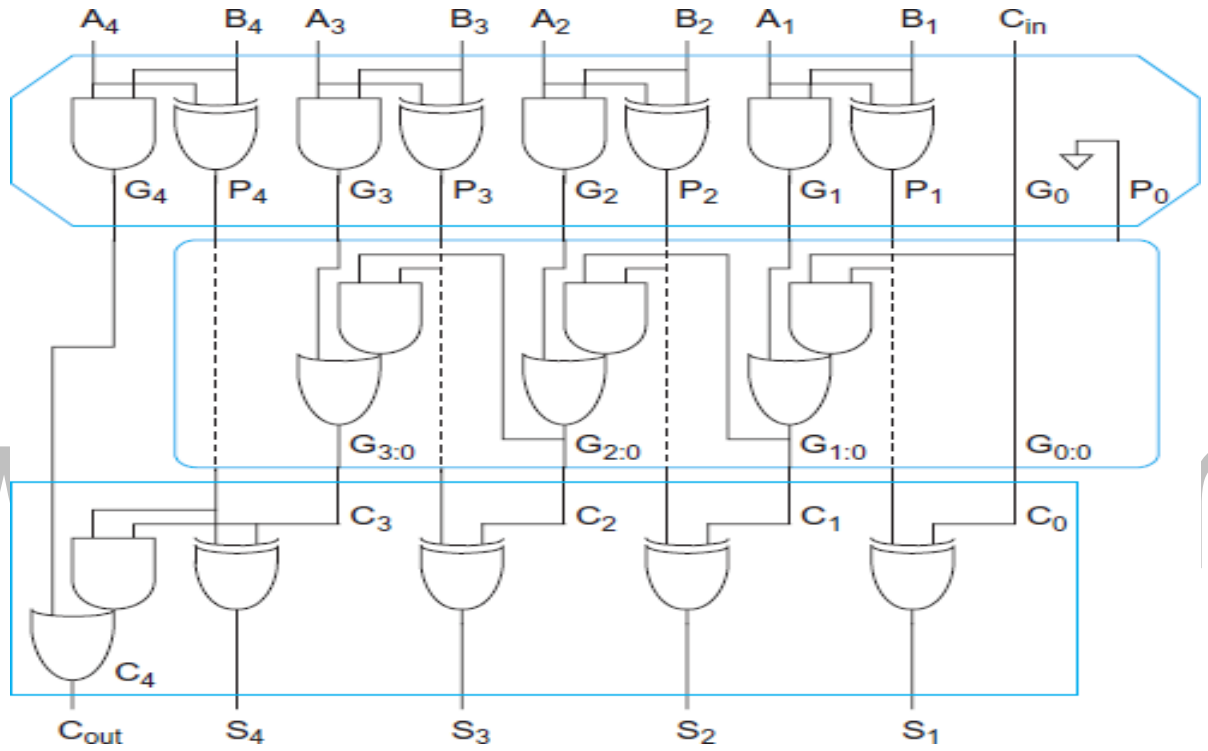


Fig.4.1.4 (d) 4-bit ripple carry adder

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

Fig.4.1.4 (d) shows a 4-bit carry-ripple adder. The critical carry path now proceeds through a chain of AND-OR gates rather than a chain of majority gates.

Carry Look ahead Adder

The carry-look ahead adder (CLA) computes group generate signals as well as

$$C_{o,k} = f(A_k, B_k, C_{o,k-1}) = G_k + P_k C_{o,k-1}$$

Group propagate signals to avoid waiting for a ripple to determine if the first group generates a carry. The dependency between $C_{o,k}$ and $C_{o,k-1}$ can be eliminated by expanding $C_{o,k-1}$

$$C_{o,k} = G_k + P_k(G_{k-1} + P_{k-1}C_{o,k-2})$$

In expanded form,

$$C_{o,k} = \ldots + P_k(\ldots + P_1(G_0 + P_0 C_{i,o}))$$

Here $C_{i,o} = 0$. For every bit, the carry and sum outputs are independent of the previous bits. The ripple effect has thus been effectively eliminated and therefore the addition time should be independent of number of bits. Fig.4.3 (a) Shows the carry-look ahead adder.

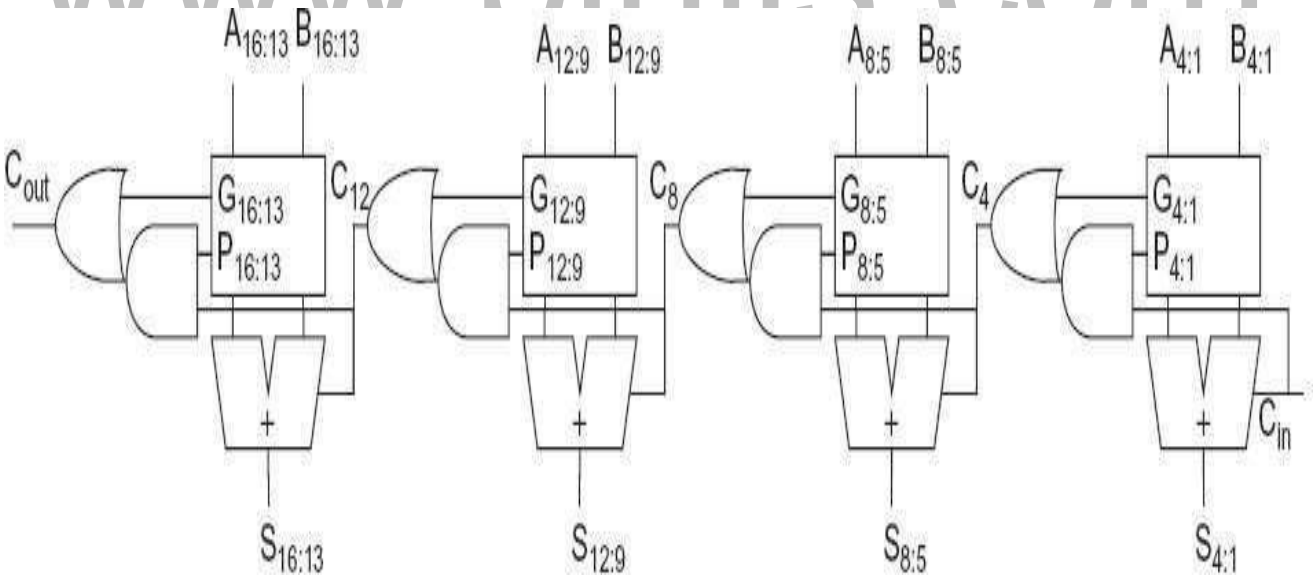


Fig.4.5.1 (a) Carry-look ahead adder

[Source: Jan M. Rabies, Samantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits: A Design perspective]

The possible circuit implementation of equation (1) is shown in Fig.4.3 (b) for N=4. The large fan-in of the circuit makes it slow for larger values of N. Implementing it with simpler gates requires multiple-logic levels. In both cases, the propagation delay increases. Furthermore, the fan-out of some signals tend to grow excessively, slowing down the ladder more since the propagation delay of a gate is proportional to its load. Finally the area of implementation grows

$$t_{\text{cla}} = t_{pg} + t_{pg(n)} + \left[\left(n-1\right)+\left(k-1\right)\right] t_{AO} + t_{\text{xor}}$$

Progressively with N. In general, a CLA using k groups of n bits each has a delay of, where tpg.(n) is the delay of the AND-OR-AND-OR-…-AND-OR gate computing the valence-n generate signal. It requires the extra n-bit generate gate, so the simple CLA is not a good design choice. CLAs often use higher-valence cells to reduce the delay of the n-bit additions by computing the carries in parallel. Fig.4.3 (c) shows such a CLA in which the 4-bit adders are built using Manchester carry chains or multiple static gates operating in parallel.

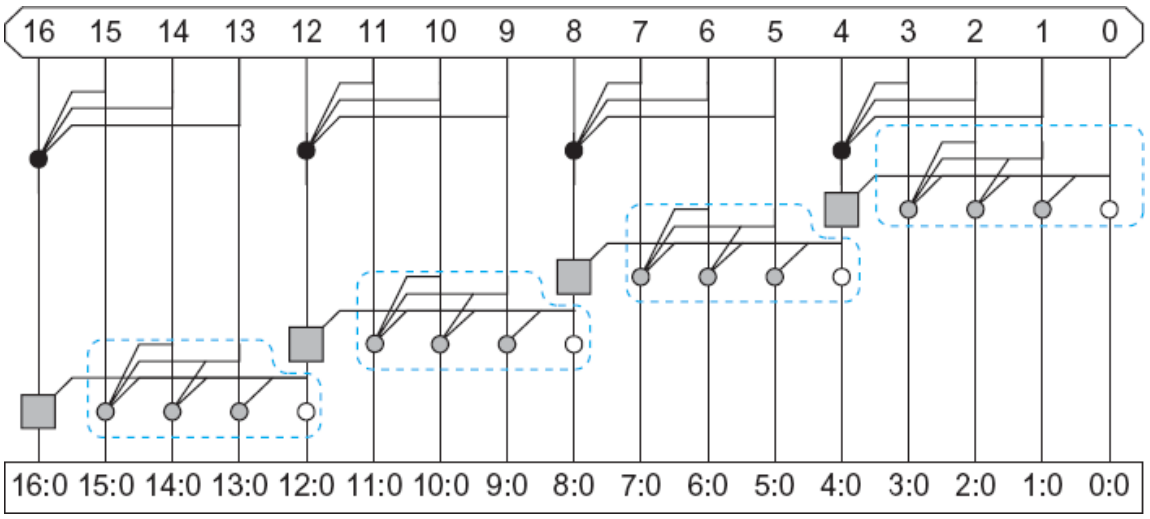


Fig.4.5.2 (b) Improved CLA group PG network

[Source: Jan M. Rabies, Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits: A Design perspective]

Memory Architecture

Memory Classification:

Read-Write Memory		Non-Volatile Read-Write Memory	Read-Only Memory
Random Access	Non-Random Access	EPROM E²PROM FLASH	Mask-Programmed Programmable (PROM)
SRAM DRAM	FIFO LIFO Shift Register CAM		

STATIC (SRAM):

- Data stored as long as supply is applied
- Large (6 transistors/cell)
- Fast
- Differential

DYNAMIC (DRAM):

- Periodic refresh required
- Small (1-3 transistors/cell)
- Slower
- Single Ended

Memory Architecture: Decoders:

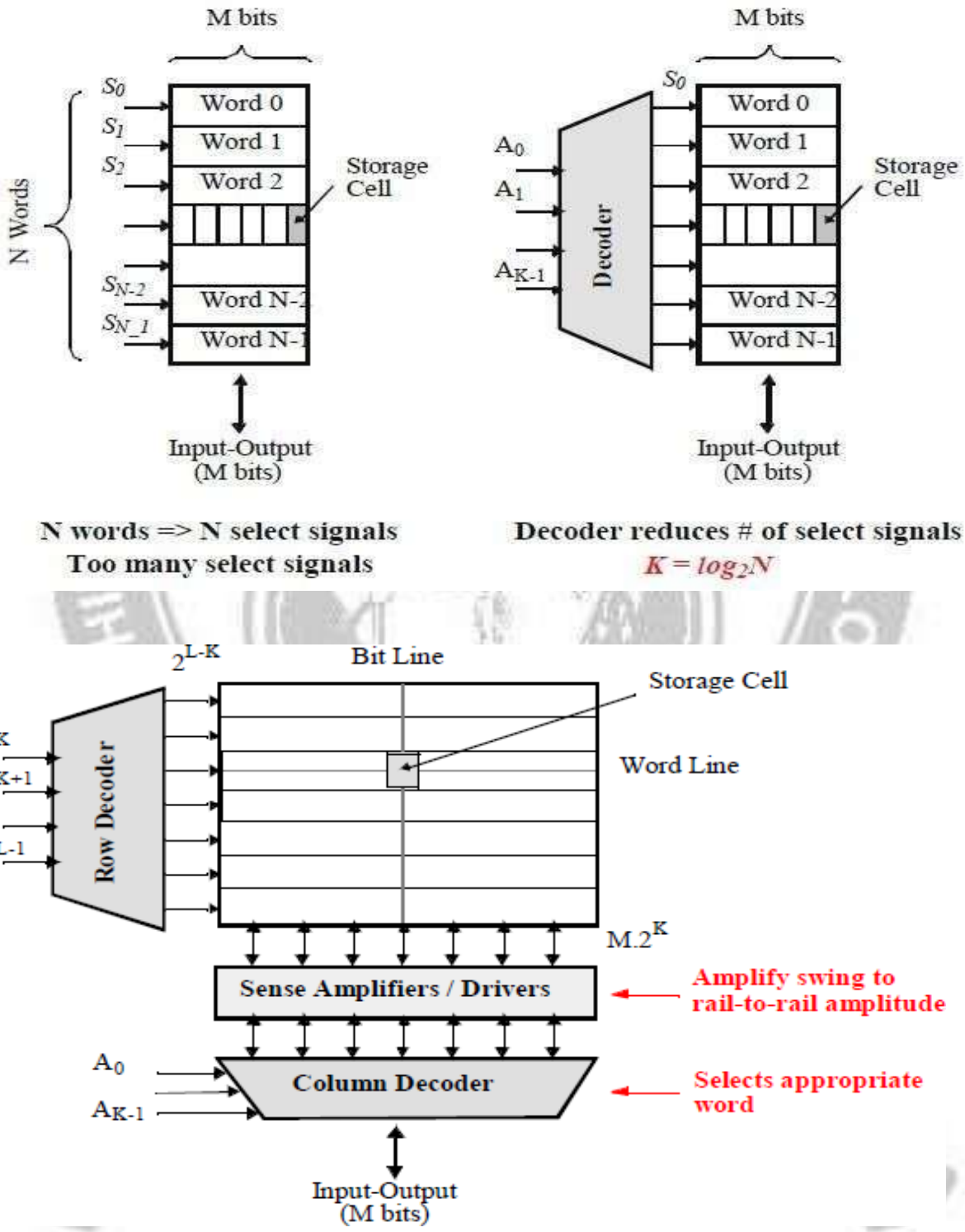


Fig 4.7.1: Array-Structured Memory Architecture:

[Source: Sung-Mo kang, Yusuf leblebici, Chulwoo Kim —CMOS Digital Integrated Circuits: Analysis & Design

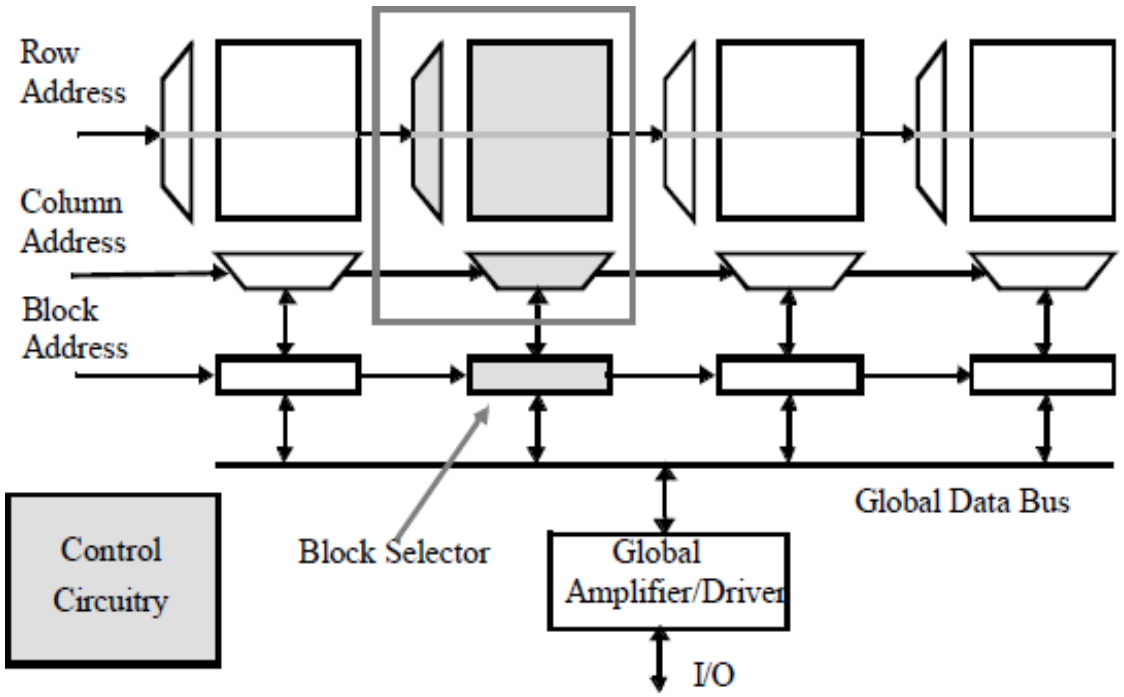


Fig 4.7.2: Hierarchical Memory Architecture

[Source: Sung-Mo kang, Yusuf leblebici, Chulwoo Kim —CMOS Digital Integrated Circuits: Analysis & Design]

Advantages:

- Shorter wires within blocks
- Block address activates only one block hence, power savings.

Tradeoffs

1) Speed, area and power can be trade off through the choice of the supply voltages, transistor threshold and device sizes.
2) Some design techniques are implemented at design time.
3) Transistor width s and lengths can be fixed at the time of design.
4) A reduction in supply voltage results in power savings and thus is the most attractive approach.

5) Reduced supply evenly lowers the power dissipation of all the logic gates.
6) In this approach, non –critical path having timing slack is supplied with low voltage without affecting the system performance.
7) Important design concepts: -

a) To select right structure before starting an circuit optimization.
b) Determine the critical timing path through the circuit.
c) Circuit size is not only determined by the number and size of the transistors.
d) An obscure optimization can sometimes help to get a better result.
e) Power and speed can be traded off through a choice of circuit sizing, supply voltages and transistor threshold

Different Types of Multipliers

Multiplication is less common than addition, but is still essential for microprocessors, digital signal processors, and graphics engines. The most basic form of multiplication consists of forming the product of two unsigned (positive) binary numbers. For example, the multiplication of two positive 6-bitbinary integers, 2510 and 3910, proceeds as shown in Fig.4.2.1 (a).M × N-bit multiplication P = Y × X is performed by forming N partial products of M bits each, and then summing the appropriately shifted partial products to produce an M+ N-bit result P. Binary multiplication is equivalent to a logical AND operation. Therefore, generating partial products consists of the logical Landing of the appropriate bits of the multiplier and

Multiplicand. Each column of partial products is added and the carry values are passed to the next column.

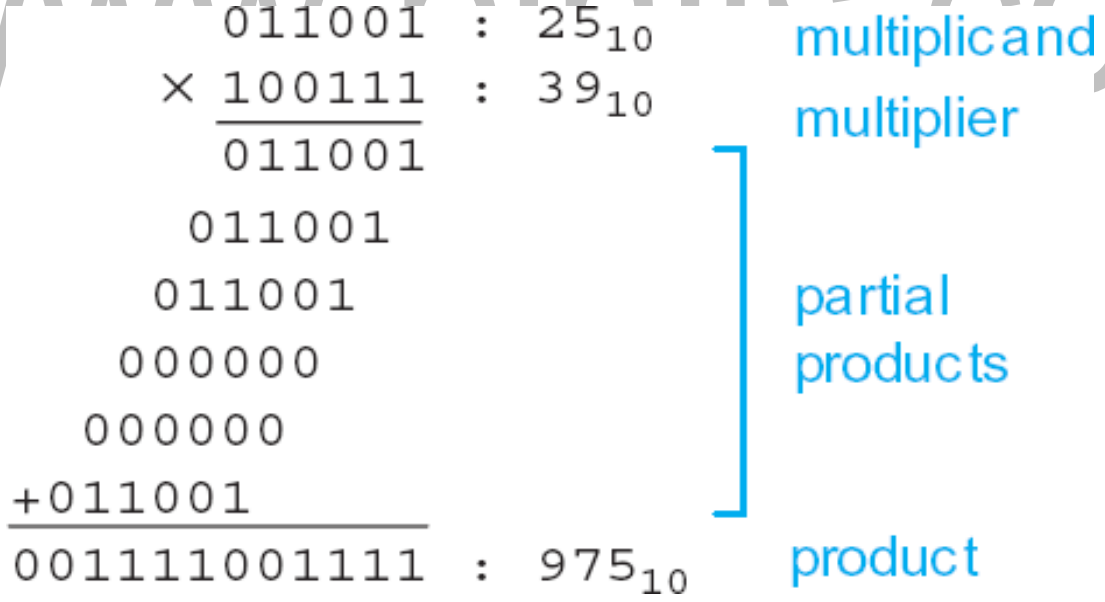


Fig.4.2.1 (a) Multiplication example

[Source: Jan M. Rabies, Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits: A Design perspective]

The multiplicand is denoted as Y = {yM —— 1, yam —— 2... y1, y0} and the multiplier is denoted as X = { xN —— 1, xN —— 2… x1, x0}. The product is given in equation (1). Fig.4.4 (b) illustrates the generation, shifting, and summing of partial products in a 6 × 6-bit multiplier. This set of operations can be mapped directly into hardware and the resulting structure is called an array multiplier.

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

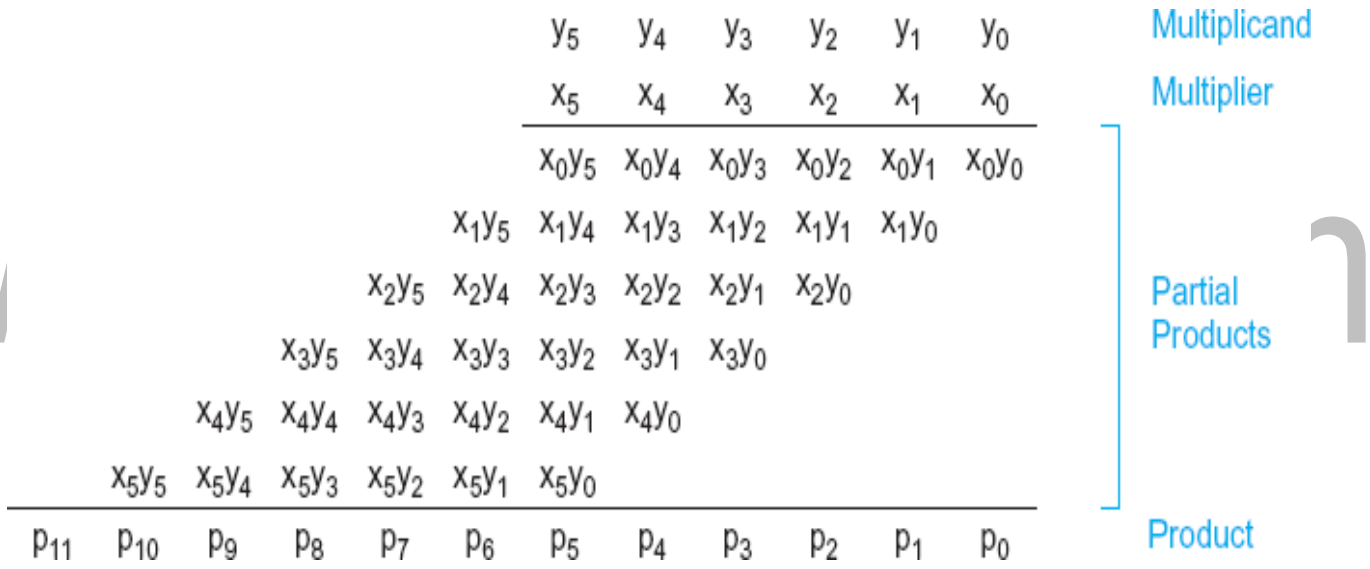


Fig.4.2.2 (b) Partial products the array Multiplier

[Source: Jan M. Rabies, Anantha Chandrakasan, Borivoje. Nickelic, ‖Digital Integrated Circuits: A Design perspective]

The composition of an array multiplier is shown in Fig.4.4 (c). There is a one-to- one correspondence between this hardware structure and the manual multiplication in Fig.4.4 (a). The generation of partial product requires a multiplication by 1 or 0 (i.e.) AND operation. Generating the N partial products requires N M-bit AND gates. The shifting of the partial products is performed by

Simple routing and does not require any active logic. The overall structure can be compacted into a rectangle, resulting in a very efficient layout.

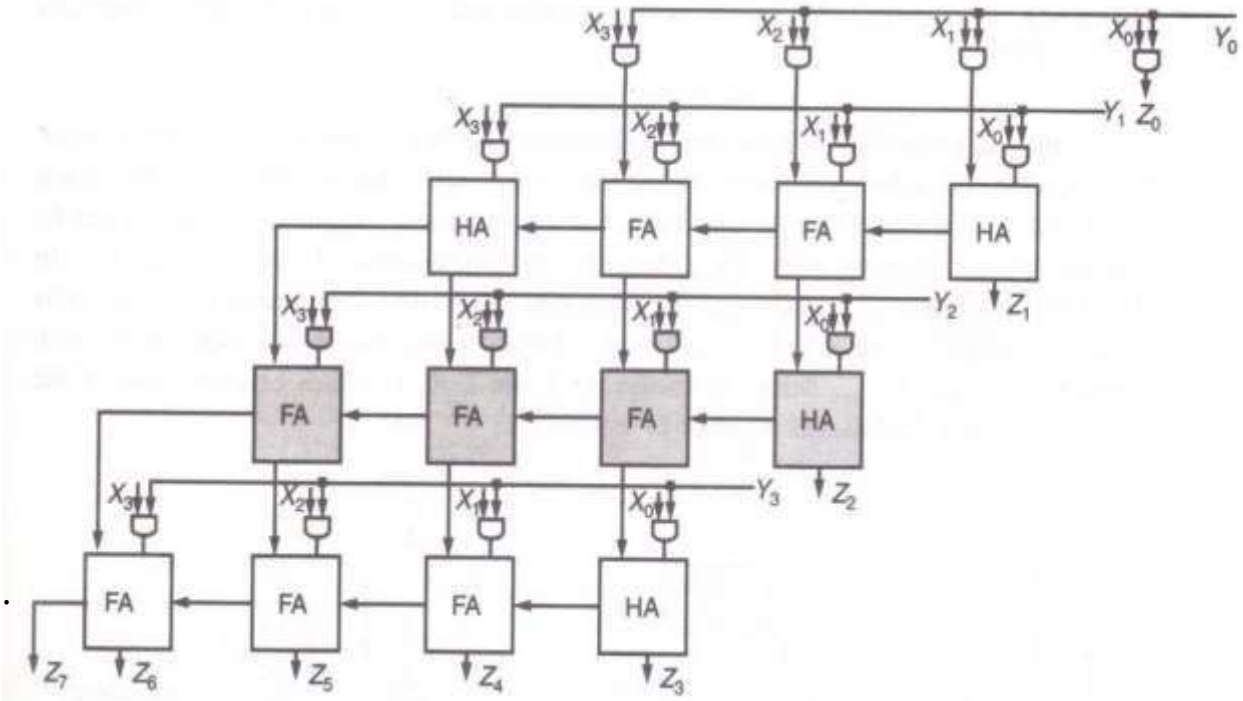


Fig.4.2.1 (c) 4x4 Multiplier for unsigned numbers

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

Due to array organization, determining the propagation delay is difficult. Partial sum adders are implemented as ripple-carry adders. Performance optimization requires critical timing path to be identified. Two such paths are highlighted in Fig.4.4 (d). The propagation delay is given as,

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

Where t_{carry} is the propagation delay between input and output carry, is the delay between the input carry and sum bit of the full adder and t_{and} is the

Delay of the AND gate. t_{sum}

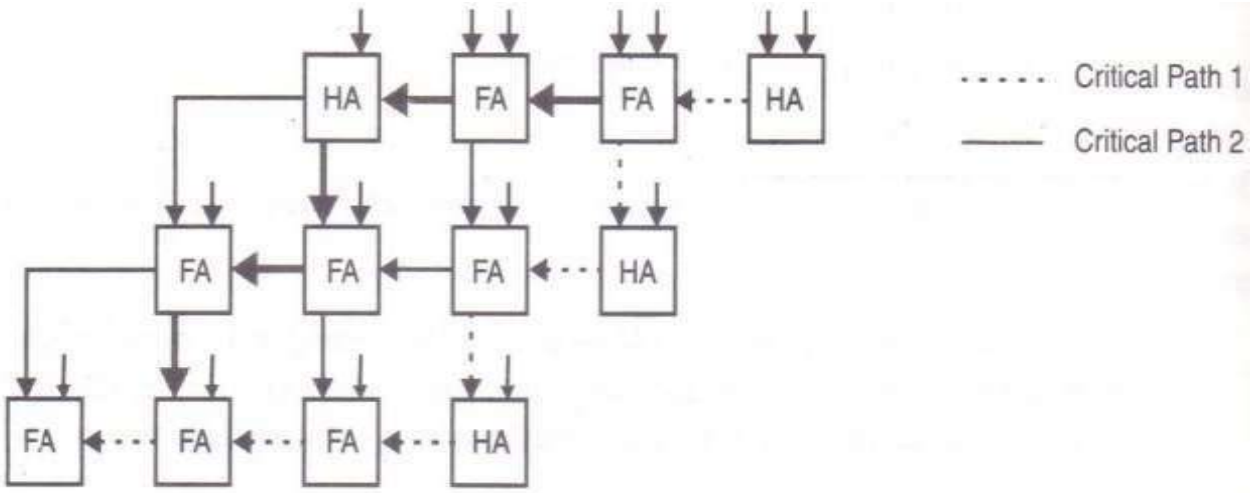


Fig.4.2.1 (d) Ripple carry based 4x4 multiplier with two critical paths highlighted

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

Since all critical paths have the same length, speeding up one of them does not make much difference. All the critical paths have to be speeded up at the same time. From equation (2), it is deduced that the minimization of requires the minimization of both

t_{carry} and. t_{sum}

Due to large number of identical critical paths, increasing the performance of the structure shown in Fig.4.4 (d) is achieved with careful transistor sizing. A more efficient multiplier structure is obtained by noticing that the multiplication result does not change when the output carry bits are passed diagonally downwards instead of to the right. An extra adder called as a vector-merging adder, is added to generate the final result. Such multiplier is called as **carry-save multiplier,** because the carry bits are not immediately added but are rather saved for the next adder stage. This structure has a slightly increased area cost but it has

The advantage that its worst-case-critical path is uniquely defined as shown in fig.4.4 (e) and expressed in equation (3).

$$1)t_{carry} + t_{and} + t_{merge} = (N - \quad (3)$$

Assuming that $t_{and} = t_{carry}$

A simple way to reduce the propagation delay of this structure is to minimize t_{merge}. This is achieved by using a fast adder implementation such as a Carry-select or a look ahead structure for the merging folder.

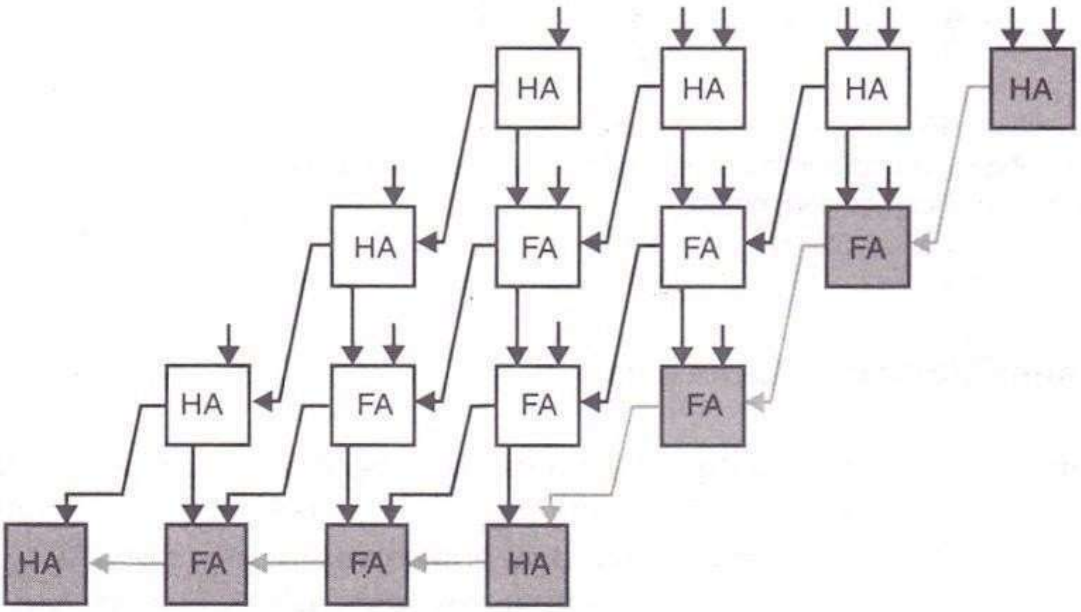


Fig.4.2.1 (e) 4x4 carry-save multiplier

[Source: Jan M. Rabaey ,Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design perspective]

Barrel shifter

A barrel shifter performs a right rotate operation. It handles left rotations using the complementary shift amount. Barrel shifters can also perform shifts when suitable masking hardware is included. Barrel shifters come in array and logarithmic forms. The logarithmic barrel shifters are most useful because they are better suited for large shifts. Fig.4.2 (a) shows a simple 4-bit barrel shifter that performs right rotations. Unlike funnel shifters, barrel shifters contain long wrap-around wires.

In a large shifter, it is necessary to upsize or buffer the drivers for these wires. Fig.4.2 (b) shows an enhanced version that can rotate left by prorogating right by 1, then rotating right by k. Performing logical or arithmetic shifts on a barrel shifter requires a way to mask outthe bits that are rotated off the end of the shifter, as shown in fig 4.2.

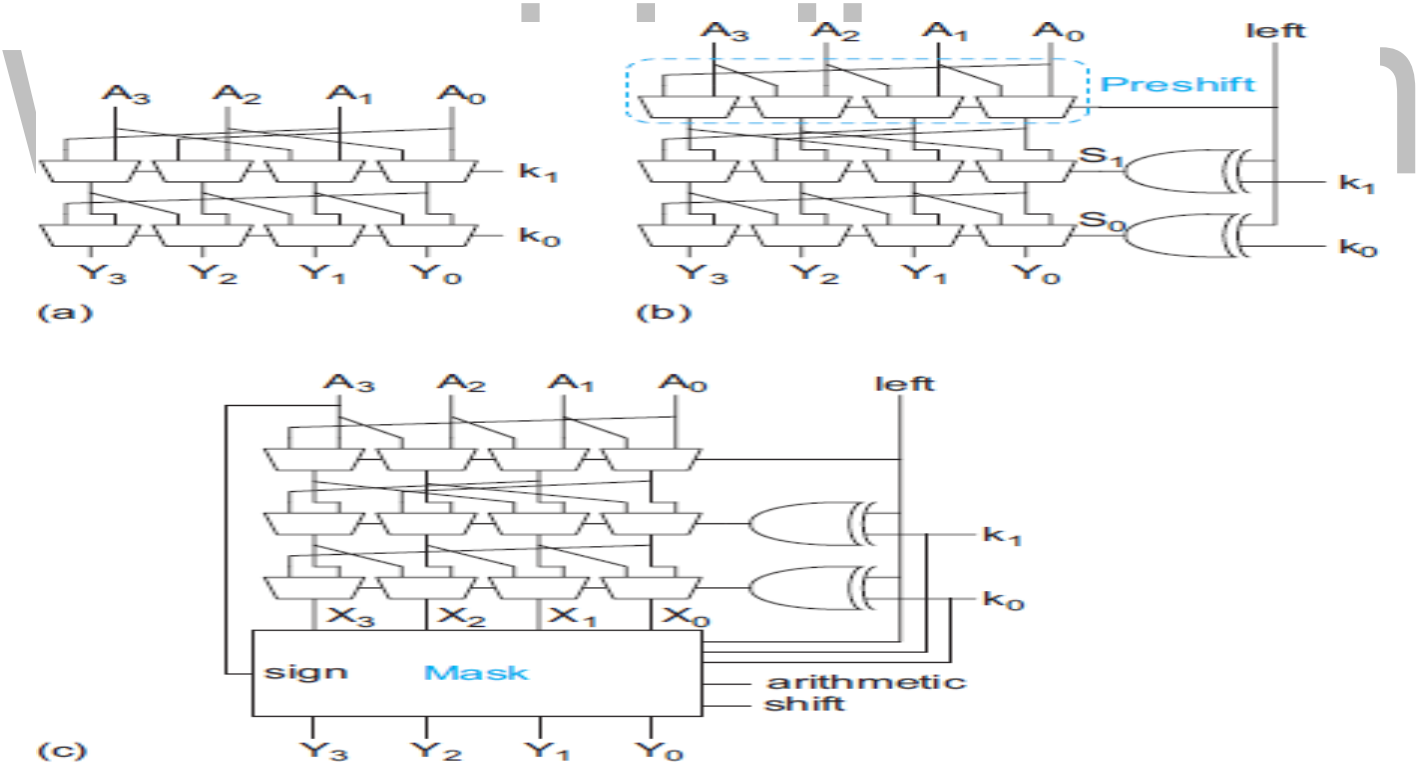


Fig.4.3.1 Barrel shifters: (a) rotate right, (b) rotate left or right, (c) rotates and shifts

[Source: Jan M. Rabaey, Anantha Chandrakasan, Borivoje. Nikolic, ‖Digital Integrated Circuits:A Design

Fig.4.3.1 shows a 32-bit barrel shifter using a 5:1 multiplexer and an 8:1 multiplexer. The first stage rotates right by 0, 1, 2, 3, or 4 bits to handle a pre-rotate of 1 bit and a fine rotate of up to 3 bits combined into one stage. The second stage rotates right by 0, 4, 8, 12, 16, 20, 24, or 28 bits. The critical path starts with decoding the shift amount for the first stage. If the shift amount is available early, the delay from A to Y improves substantially.

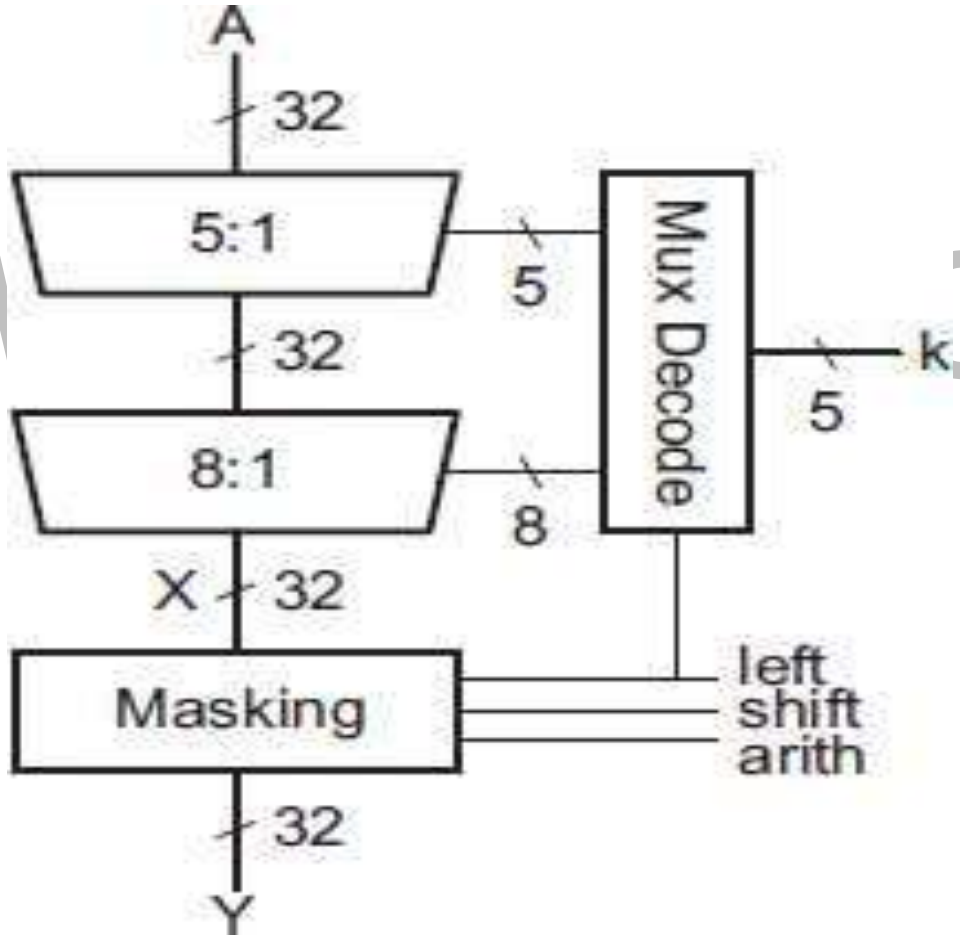
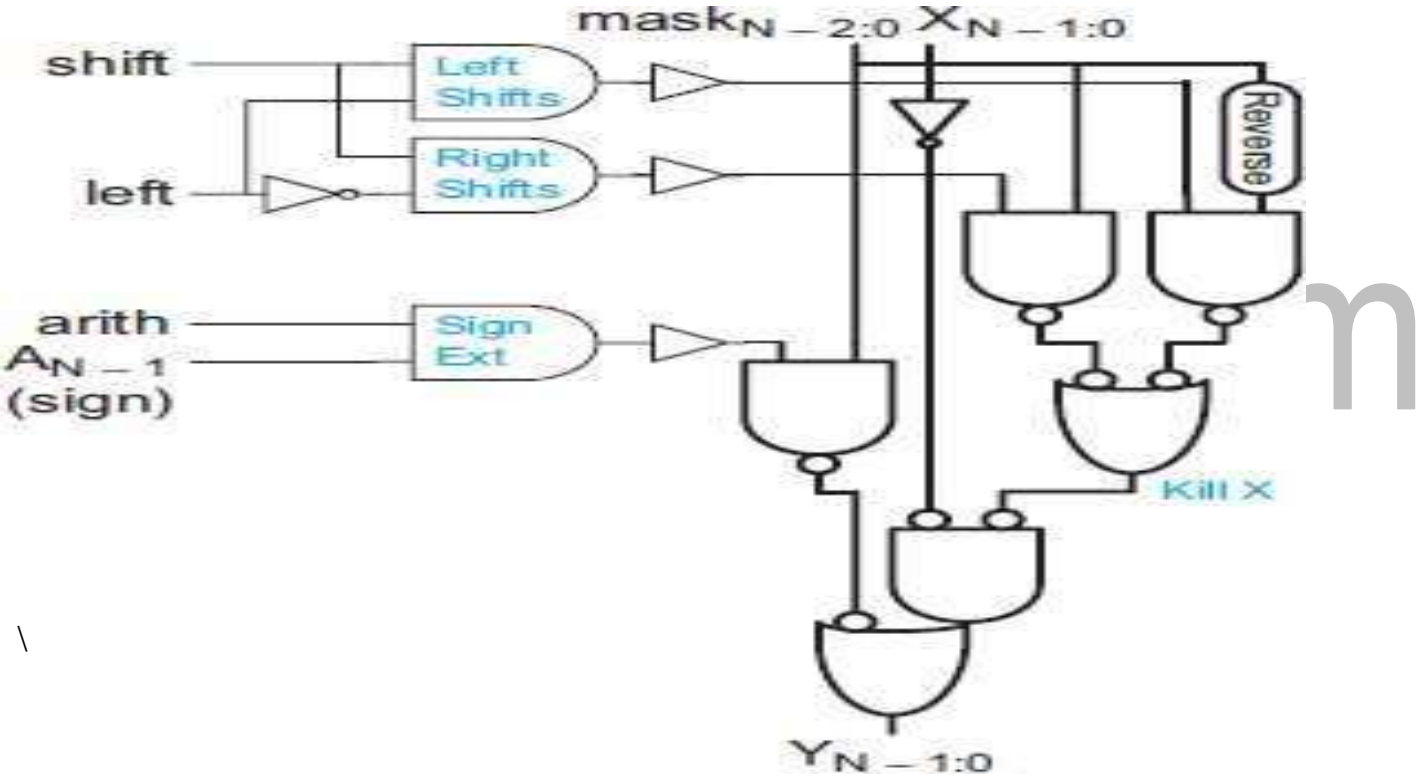


Fig.4.3.2: 32-bit logarithmic barrel shifter

[Source: Wayne Wolf, —Modern VLSI Design: System On Chip]

While the rotation is taking place, the masking unit generates an N-bit mask with ones where the kill value should be inserted for right shifts. For a right shift by m, the m most significant bits are ones. This is called a thermometer code. When the rotation result X is complete, the masking unit replaces the masked bits with the kill value. For left shifts, the mask is reversed.



\

Source: Wayne Wolf, —Modern VLSI Design: System On Chip]

Fig 4.3.3 shows masking logic. If only certain shifts are supported, the unit can be simplified, and if only rotates are supported, the masking unit can be eliminated, saving substantial hardware, power, and delay.