UNIT IV DESIGNING ARITHMETIC BUILDING BLOCKS

Data path circuits, Architectures for ripple carry adders, carry look ahead adders, High speed adders, accumulators, Multipliers, dividers, Barrel shifters, speed and area tradeoff

4.1 Introduction

Chip functions generally can be divided into the following categories:
1.Datapath operators
2.Memory elements
3.Control structures
4.Special-purpose cells

- I/O
- Power distribution
- Clock generation and distribution
- Analog and RF

CMOS system design consists of partitioning the system into subsystems of the types listed above. Many options exist that make trade-offs between speed, density, programmability, ease of design, and other variables. Data path operators benefit from the structured design principles of hierarchy, regularity, modularity, and locality. They may use N identical circuits to process N-bit data. Related data operators are placed physically adjacent to each other to reduce wire length and delay. Generally, data is arranged to flow in one direction, while control signals are introduced in a direction orthogonal to the dataflow. Common data path operators considered in this chapter include adders, one/zero detectors, comparators, counters, Boolean logic units, error-correcting code blocks, shifters, and multipliers.

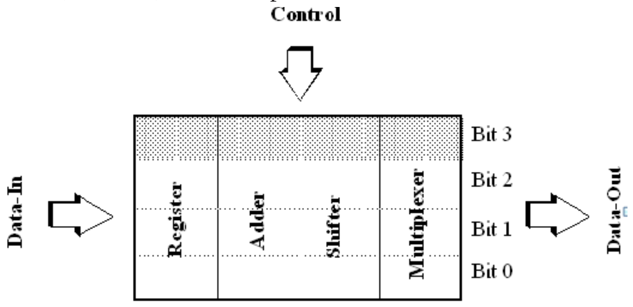


Fig.4.1:Bit siliced datapath organization

4.2.Architecture for ripple carry Adder:

Ripple carry Adder is a simplest design. cascade connection of full adders forms a ripple carry adder .In this Critical path goes from Cin to Cout. In this ripple carry adder , each of the full adder can be made up for XOR, OR AND gate. In a full adder circuit is implemented using XOR, OR and AND gate

In order to add binary numbers with more than one bit, additional full-adder must be employed. A n bit parallel adder can be constructed using number of full adder circuits connected in parallel. The block diagram of n-bit parallel adder adder using number of full adder circuits connected in cascsde ,i.e., the carry output of each adder connected to the carry input of the next higher order adder.

It should be noted that either a half-adder can be used for the least significant position or the carry input of a full-adder is made 0 because there is no carry into the least significant bit position.

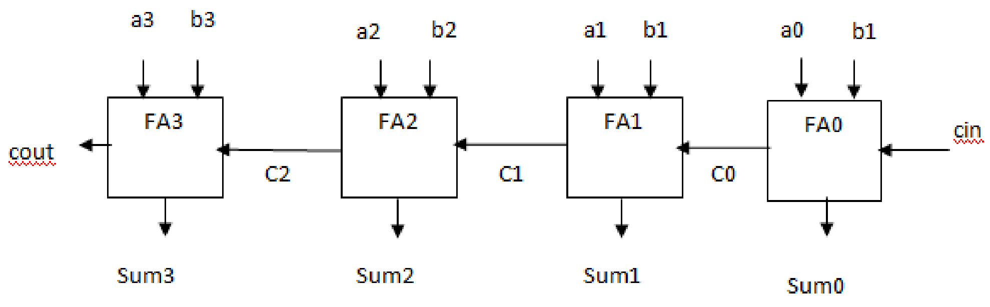


Fig.4.2:Four bit ripple carry Adder

4.3.Carry Look Ahead Adder:

The linear growth of adder carry delay with the size of the input word may be improved by calculating the carries to each stage in parallel. The carry of the ith stage , C_i may be expressed as,

Ci=Gi+Pi.CI-1
Where

Gi=Ai.Bi	generate signal
Pi=Ai+Bi	propagate signal

Expanding this yields
Ci=Gi+PiGi-1+PiPi-1Gi-2+…+Pi…P1C0
The sum Si is generated by
Si=Ci-1 xor Ai xor Bi

The size of the gates needed to implement this carry lookahead scheme can clearly get out of hand. As a result ,the number of stages of lookahead is usually limited to about four. For four stages of lookahead,, the appropriate terms are

C1=G1+P1C0
C2=G2+P2g1+P2G1+p2P1C0
C3=G3+P3G2+P3P2G1+P3P2P1C0
C4=G4+P4G3+P4P3G2+P4P3P2G1+P4P3P2P1C0.

A possible implementation of the carry gate for this kind of carry lookahead adder for 4 bits is shown in figure 8.11.Note that the gates has been partitioned to keep the number of inputs less than or equal to four. This typical of the type of carry lookahead that would be used in a gate array or standard cell design. The circuit and layout are quite irregular. Taking the term of C4,we note that it may be expressed as

C4=G4+P4.(G3+P3.(G2+P2.(G1+P1C0)))

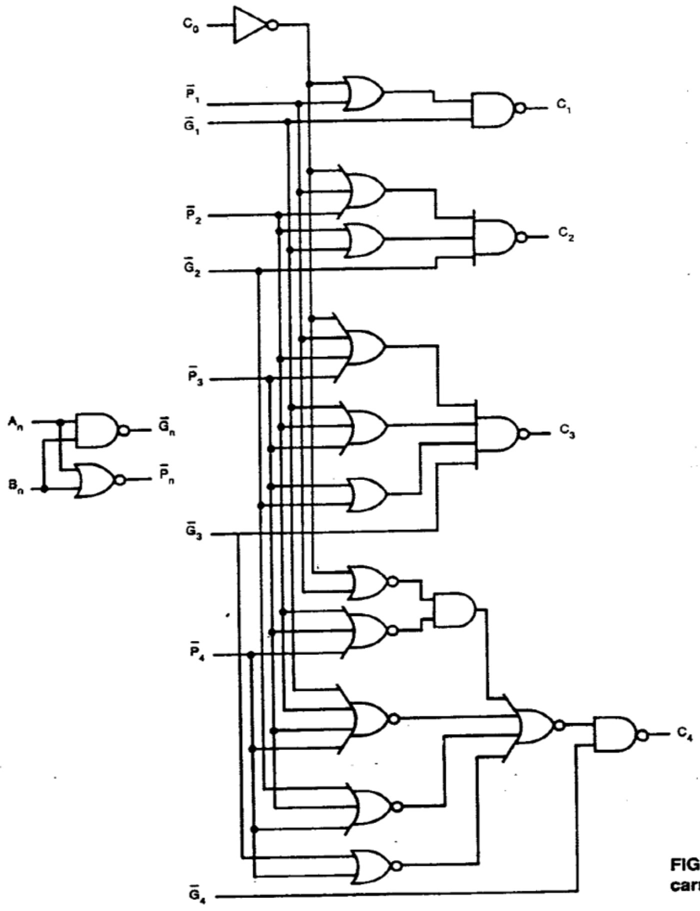


FIGURE 8.11. **4-bit full carry lookahead stage**

4.3.1.Binary lookahead Adder:

Reviewing the equation for the binary adder we have

Ci=Gi+PiCi-1
Pi=Ai+Bi or Ai xor Bi
Gi=Ai.Bi
Si=Ci-1 xor Pi (if Pi=Ai xor Bi)

Both Gi and Pi can be determined in constant time, so Ci is the only time critical term that needs to be calculated. We can define a new operator O, which has the following functions:

(g,p) o(g’,p’)=(g+(p.g’),p.p’)

Where g,p,g’,p’ are Boolean variables. It can be shown that this new operator is associate and the carry signals can be determined by

Ci=Gi

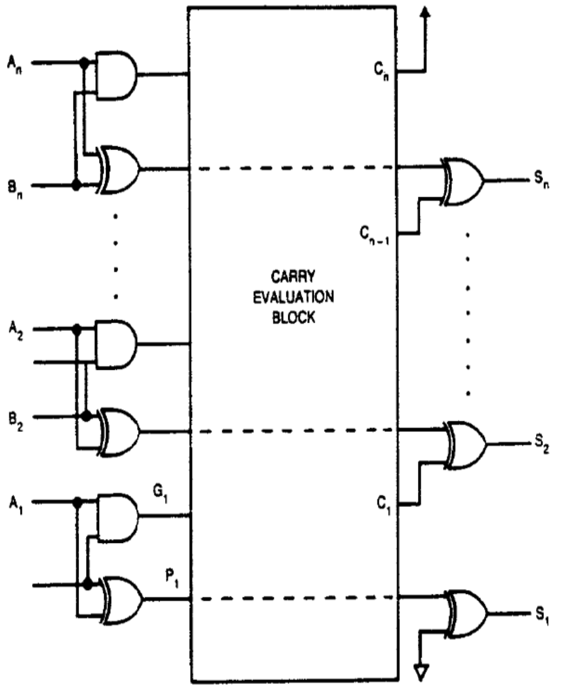


FIGURE 8.17. Carry look-ahead adder

Where

$$(Gi,Pi)=\begin{cases} (g1, p1) & \text{if } i = 1 \\ (gi, pi) \ldots o \ldots (Gi-1, Pi-1) & \text{if } 2 \le i \le n \\ = (gi, pi)\ o\ (gi-1, pi-1) \ldots o \ldots (g1, p1) \end{cases}$$

The associate property of the o operator allows the processing elements to be embedded in a binary tree structure of depth O(logn). A generalized carry lookahead adder is shown in fig.8.17.It is composed of a G and p term generator, the carry bock and a sum block.

4.4.High Speed Adders (HSA):

All High Speed Adders (HSA) have the objective of decreasing the computation time and different tradeoffs. This paper examines few of them bellow.

4.4.1 Carry Skip Adder

A carry-skip adder is designed to speed up a wide adder by adding the propagation of carry bit around a portion of the entire adder. The idea is illustrated in figure 5 for the case of a 4 bit adder. The carry-in bit is designated as Ci and the adder itself produces a carry-out bit of

Ci+4. The carry skip circuitry consists of two logic gates. The AND gate accepts the carry-in bit and compares it to the group propagate signals.

$$P_{(1,i+3)}=P_{i+3},P_{i+2},P_{i+1},P_i$$

Using the individual propagate values, the output from the AND gate is ORed with Ci+4 to produce a stage output of As shown in the figure 5, if P(i, i+3) = 0, then the carry-out of the group is determined by the value of Ci+4. However, if P(i,i+3) = 1, then the carry-in bit is Ci= 1, then the group carry-in is automatically send to the next group of adders. The name "carry-skip" is due to the fact that if the condition P(i,i+3). Ci is true and then the carry-in bit skips the block entirely.

$$C_{arry} = C_{i+4}+P_{(i,i+3).}C_i$$

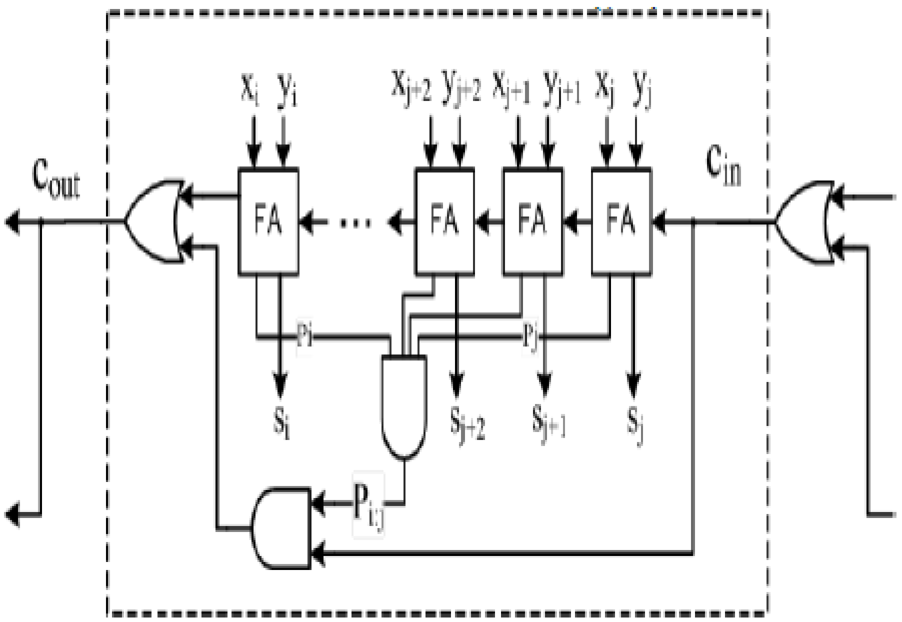
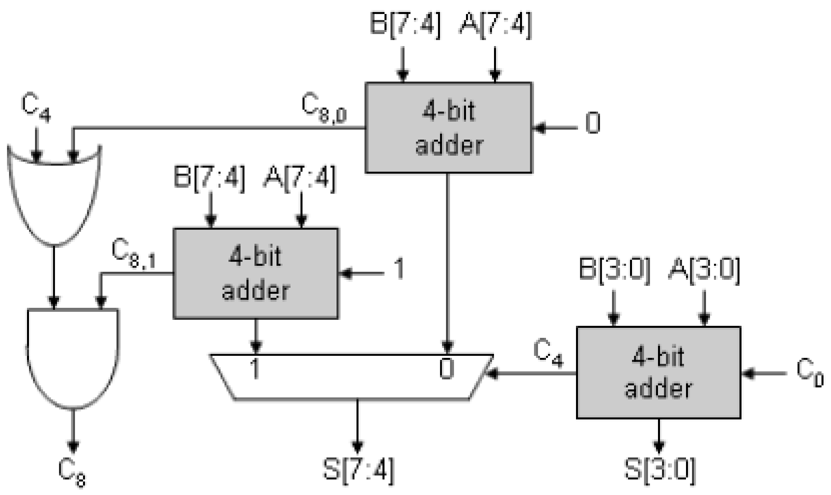


Figure 5. Carry skip adder

<u>4.4.2 Carry – Select Adder</u>

Carry Select Adders (CSA) use multiple narrow adders to create fast wide adders. Consider the addition of two n bit numbers with a = an-1…..a0, and b = bn-1…..b0. At the bit level the adder delay increases from the least significant 0th position upward, with the (n-1)th requiring the most complex logic. A carry select adder breaks the addition problem into smaller groups. A carry-select adder provides two separate adders for the upper words, one for each possibility. A multiplexer (MUX) is then used to select the valid result. The figure 6 shows the block diagram of CSA. As a concrete example, consider an 8-bit adder that is split into two 4-bit groups. The lower order bits a3 a2 a1 a0 and b3 b2 b1 b0 are fed into the 4-bit adder to produce the sum bits S3 S2 S1S0 and a carry-out bit C4 as shown

Figure 6. Carry Select Adder



The higher order bits a7 a6 a5 a4 and b7 b6 b5 b4 are used as two 4-bit adders. Adder calculates the sum with a carry in of C=0, while the other adder does the same only it has a carry-in value of C=1. Both sets of results are used as inputs to an array of 2:1 MUXs. The carry bit C4 from the first adder is used as the select signal to MUX. If C4 = 0, then the result of C=0 adder are sent to the output, while a value of C4=1 selects the result of C=1 adder for S7 S6 S5 S4. The carry-out bit C8 is also selected by the MUX array. The design speeds up the addition of the word by allowing the upper and lower portions of the sum to be calculated simultaneously. The price paid is that it requires an additional word adder, a set of multiplexers and associated interconnect wiring. The design becomes viable if speed is more important than area consumption.

4.4.3 Carry – Save Adder

Carry – save adder are based on the idea that a full adder really has three inputs and produces two outputs as shown. While it is usually associates the third input with a carry in, it could equally well be used as a "regular" value. The full adder is used as 3:2 reduction network, where it starts with bits from 3 bits words, adds them and then has an output that is 2-bits wide. An n-bit carry save adder can be build by using n separate adders. The name „carry-save" arises from the fact that we save the carry out words instead of using it immediately to calculate the final sum. Carry-save adders are useful in situations where we need to add more than two numbers. Since the design automatically avoids the delay in the carry-out bits.

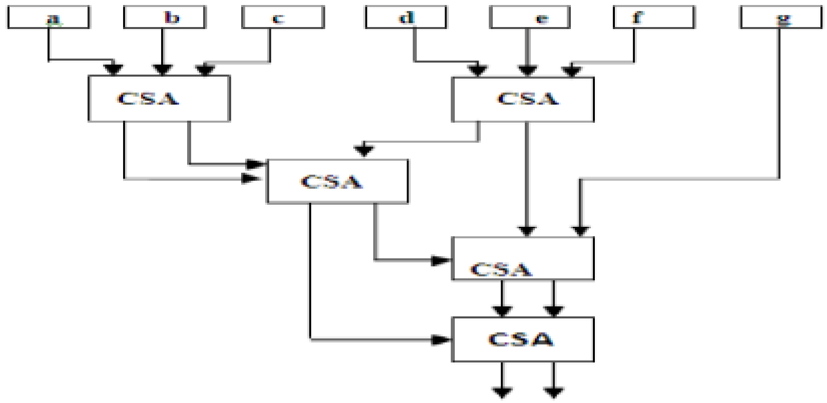


Figure 7. Carry save Adder

4.5.Accumulator Basics

An accumulator is build with an adder whose sum can be loaded into a register as shown in figure 1. Accumulators are a basic building block of most large digital logic or DSP project. As an analogy, you can think of an up accumulator (the type we are using in this project) as a file cabinet. It starts out empty. If you add two, it now holds the value of two. If you add three more it now holds five.

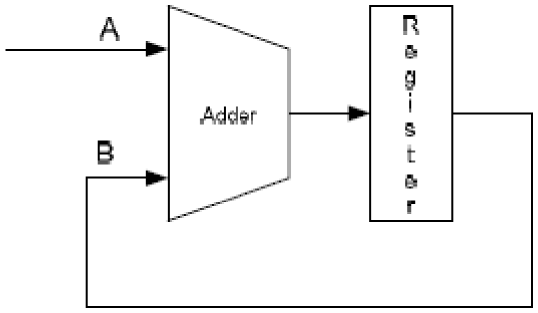


Figure 1; Block diagram for an Accumulator

Figure 4.15 shows a single bit accumulator and figure 4.16 shows an n-bit accumulator.

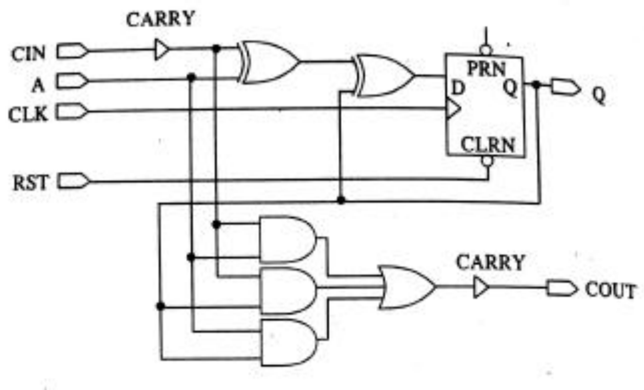


Figure 4.15 Single Bit Accumulator

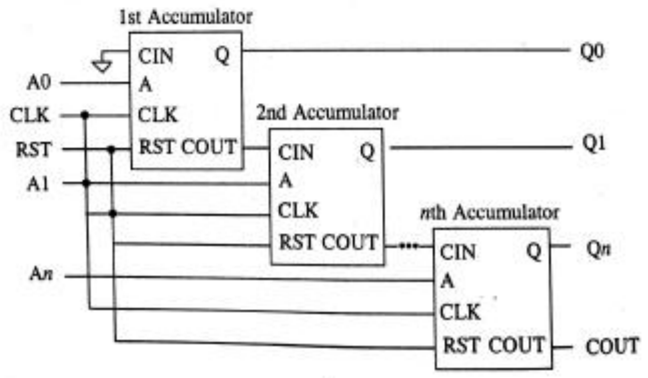


Fig.4.15:n bit Accumulator

4.6. Multipliers:

In many signal processing operation, such as correlations, convolution, filtering, and frequency analysis. one need to perform multiplication. We will use the multiplication algorithms to illustrate methods of designing different cells so that they fit into a larger structure. In order to introduce these designs, we will briefly introduce simple serial and parallel multipliers. The most basic form of multiplication consists of forming the product of two positive binary numbers. The multiplication of two positive binary integers 12_{10} and 5_{10} may proceed in the following manner.

```
Multiplicand: 1100          : 12₁₀
 Multiplier:  0101          : 5₁₀
              ---------
               1100
               0000
               1100
               0000
```

0111100 :60_{10}

Therefore multiplication process may be viewed to consist of the following two steps:

1. Evaluation of partial product
2. Accumulation of the shifted partial product.

It should be noted that binary multiplication is equivalent to a logical AND operation. Thus evaluation of partial products consists of the logical AND ing of the multiplicand and the relevant multiplier bit.There are number of techniques that may be used to perform multipliction.In general,the choice is based upon factors such as speed,throughput,numerical accuracy, and area.As a rule ,multipliers may be classified by the format in which data words are accessed namely,

1.Serial form
2.Serial/Parallel form
3.Parallel form

4.6.1.Serial Multiplier**:**

The simplest form of serial multiplier shown in fig.8.30 uses the successive addition algorithm and is implemented using a full adder, a logical AND circuit a delay element (i.e., either static or dynamic flip-flop) and a serial to parallel register.

The two numbers X and Y are presented serially to the circuit.The partial product is evaluated for every bit of the multiplier and a serial addition is performed with the partial additions already stored in the register.The AND gate G2 between the input to the adder and the output of the register is used to reset the partial sum at the beginning of the multiplication cycle.If the register is made of N-1 stages,then the 1-bit shift required for each partial product is obtained automatically.As far as the speed of operation is concerned the complete product of M+N bits can be obtained in M+N intervals of the multiplicand clock.

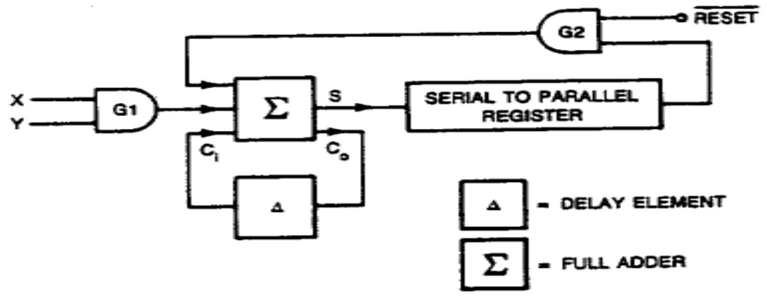


Fig.8.30: Serial Multiplier

4.6.2.Serial/Parallel Multipliers:

Using this general approach,the multiplication is performed by mean of successive additions of columns of t he shifted partial product matrix.As left shifting by one bit in serial systems is obtained by a 1-bit delay element,the multiplier is successively shifted and gates the approproate bit of the multiplicand.The delayed gated instances of the multiplicand must all be in the same column of the shifted partial product matrix.They are then added to form the required product bit for the particular column.

This structure requires M+N clock cycles to produce a product.The main limitation is that the maximum frequency is limited by the propagation through the array of adders.

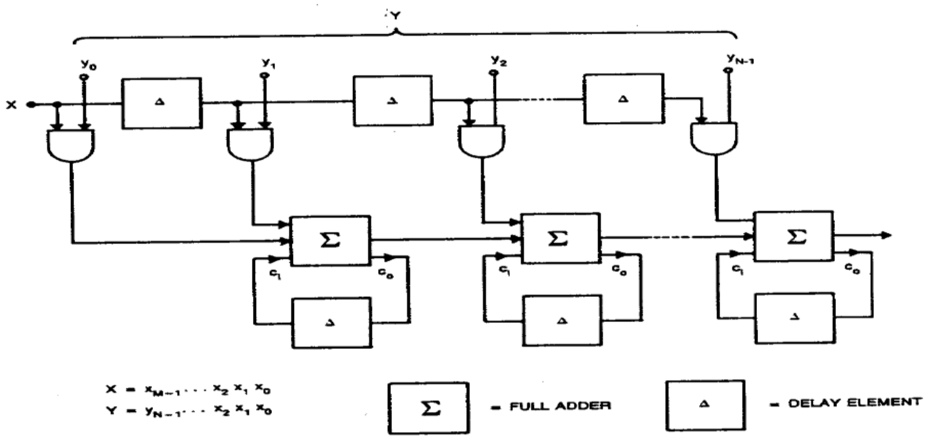

Y
y_0
y_1
y_2
y_{N-1}
X
Δ
Σ
c_i
c_o
$X = x_{M-1} \cdots x_2\, x_1\, x_0$
$Y = y_{N-1} \cdots x_2\, x_1\, x_0$
Σ - FULL ADDER
Δ - DELAY ELEMENT

Fig.8.31: Serial/Parallel Multipliers

<u>4.6.3.Parallel Multiplier:</u>

A parallel multiplier is based on the observation that partial products in the multiplication process can be independently computed in parallel.For example,consider the unsigned integers X and Y.

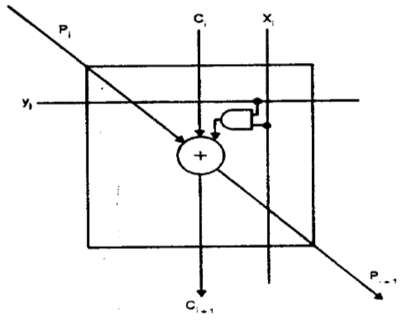
$$X=\sum_{i=0}^{m-1} X_i 2^i$$
$$Y=\sum_{j=0}^{n-1} Y_i 2^j$$

The product is found by

$$P_r=X_yY_r=\sum_{i=0}^{m-1} X_i 2^i \sum_{j=0}^{n-1} Y_i 2^j$$
$$=\sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (X_i Y_j) 2^{i-j}$$
$$=\sum_{k=0}^{m-n-1} P_k 2^k$$

Thus Pk are the partial product terms called summands.There are mn summands.Which are produced in parallel by a set of mn AND gates.Fig 8.35 shows a cell that may be used to construct a parallel multiplier.The X1 term is propagated vertically,While the Y1 term is propagated horizontally.Incoming partial partial products enter at the top left.Incoming CARRY

IN values enter at the top of the cell.The bit wise AND is performed in the cell,and the SUM is passed to the cell at the lower right.The CARRY OUT is passed to the buttom of the cell.Fig 8.35 shows the multiplier array with the partial products enumerated.



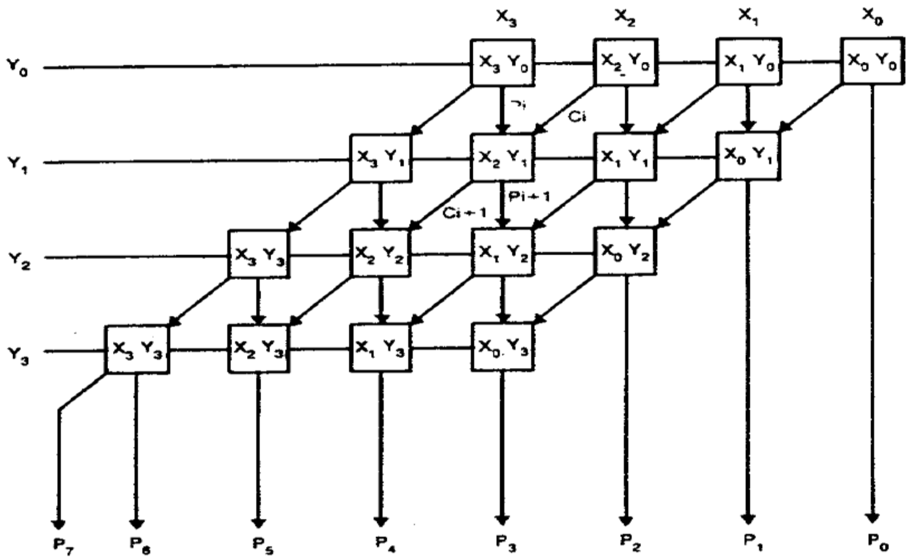


Fig.8.35: Parallel Multiplier

Fast multiplication is done similar to manually computing a multiplication operation.All the partial products are generated at the same time and organised in an array.To compute the final products a multioperand addition is used as shown in figure.This stucture is called an array multiplier.The array multiplier has the following three functions.

1.Partial product generation
2.Partial product Accumulation
3.Final Addition

EX:

```
              1  0  1  0  1  0      Multiplicand
 x                  1  0  1  1      Multiplier
 --------------------------------
              1  0  1  0  1  0   }
           1  0  1  0  1  0      }
        0  0  0  0  0  0         }  Partial products
 +   1  0  1  0  1  0            }
 --------------------------------
     1  1  1  0  0  1  1  1  0      Result
```

4.6.1. Partial product generation:

Fig shows partial product generation logic.Partial products result from the logical AND of multiplicand X with a multiplier bit Yi.Each row in the partial product array is either a copy of the multiplicand or a row of zeros.Partial product generation op[timization can lead to reduced delay and area.But partial product array has many zero rows that have no impact on the result.If multiplier consists of all ones,all the partial products exist,while in the case of all zeros,there is none.This reduces the number of generated partial products by half.

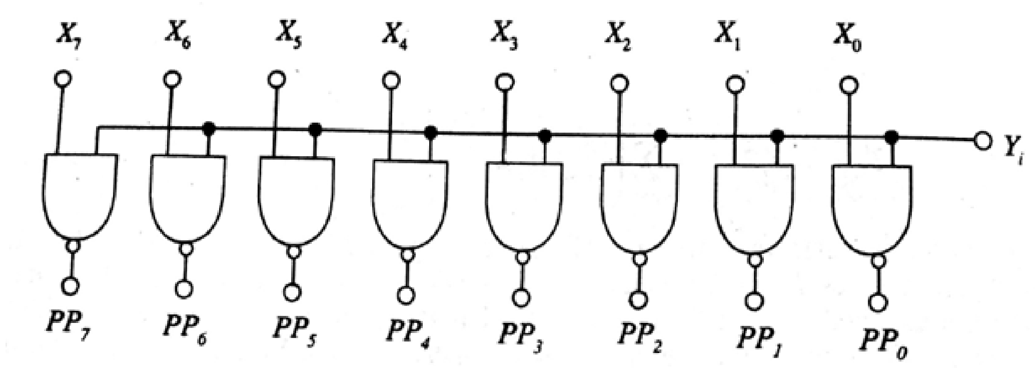


Fig.4.4: Partial product generation

Consider an 8-bit multiplier of the form 01111110, which produces six nonzero partial product rows.Reduce the number of nonzero row rows by recoding this number($2^7+2^6+2^5+2^4+2^3+2^2$) into a different format.The reader can verify that the form 10000010.1 ia s short notation for -1.This format needs to add only two partial products,but the final adder has to perform subtraction as well.This type of transformation is called Booth's recording.Booth's recording reduce the number of pertial product to at most one half.This ensures that for every two consecutive bits,at most one bit will be 1 or -1.Reducing the number of partial products is equivalent to reducing the number of additions,that speedup the operation and reduces the area.This transfermation is equivalent to formating the multiplier word into a base -4 scheme instead of the binary format.The format is,

$$Y=\sum_{j=0}^{(N-1)/2} Y_j 4^j \quad \text{with}(Y_j \in\{-2,-1,0,1,2\})$$

The 1010…10 represents the worst case multiplier input because it generates the most partial products (only half).Multiplication with {0,1} is equivalent to an AND operation multiplying with {-2,-1,0,1,2}.This requires a combination of inversion and shift logic.

In modified Booth's recording the multiplier is partitiones into 3-bit groups that overlap by one bit..The number of partial products equals half of the multiplier width.The input bits to the recording process are two current bits combined with the upper bit from the next group,moving from most significant bit to least significant bit.

4.6.2. Partial product Accumulation:

Once the partial products are generated,it has to be summed.This accumulation is a multioperand addition.This is done by using a number of adders that will form an array,called array multiplier.The following three methods are used for this accumulation.

1.Array Multiplier
2.Carry save Multiplier
3.Tree Multiplier

4.6.2.1. Array Multiplier**:**

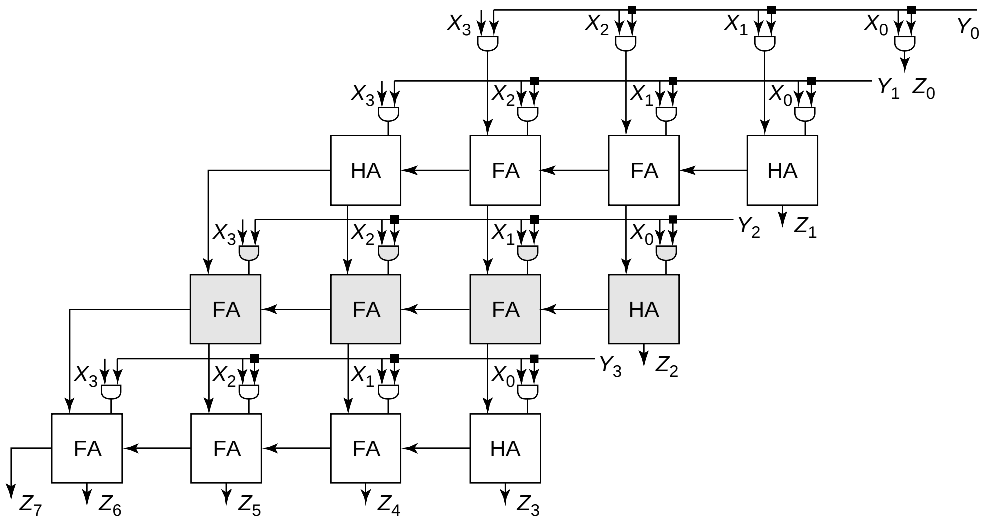


Fig.4.5: Array Multiplier

Fig. shows a 4*4 bit array multiplier.Generation of N partial products reqquires N*M 2-bit AND gates.Multipliers most area is utilised to add the N partial products.This requires N-1 M-bit adders.Shifting of the partial products for proper alignment is performed by simple routing and does not require any logic.The overall structure is compacted into a rectangle to get a very efficient layout.FA represents full adder, and half adder or an adder with two inputs.

Propagation delay in an array organizes circuit is difficult to find.The partial sum adders are implemented using ripple carry structures as shown in fig 4.6.Performance optimization requires that the critical timing path be identified first.This turns out to be nontrivial.A large number of paths of almost identical length is identified as highlighted in fig.4.6.The critical paths yields an approximate expression for the propagation delay as,

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

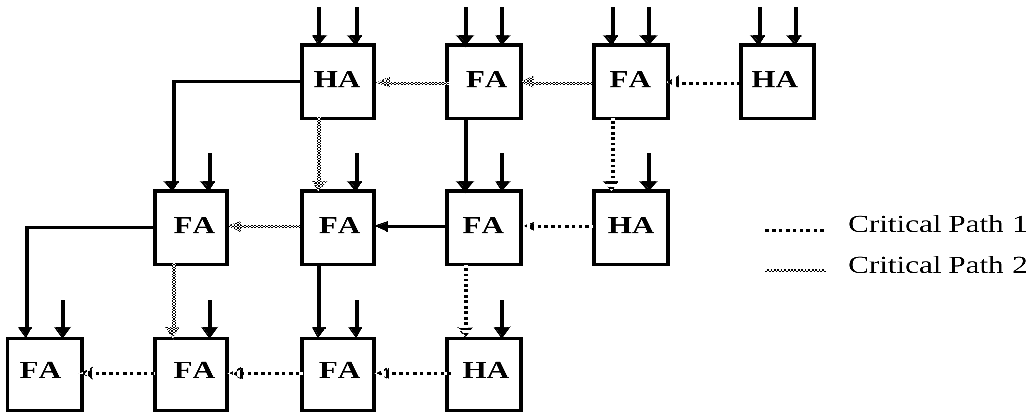


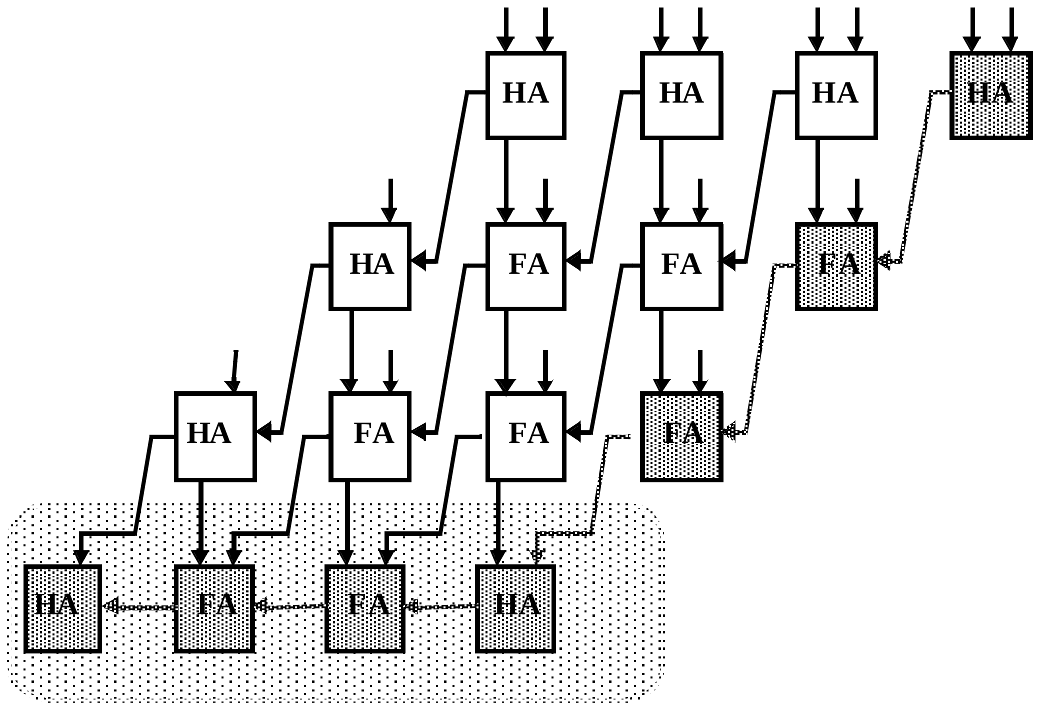
Fig.4.6:The partial sum adders are implemented using ripple carry structures

All critical paths have same length. Speeding up one of them for instance or by replacing one adder by a faster one such as a carry select adder does not make much sense. All critical paths have to be attacked at the same time .Minimization of t_{mult} requires the minimization of both tcarry and t_{sum}.

4.6.2.2. Carry Save Multiplier:

A more efficient realization is obtained by noticing that the multiplication result does not change when the output carry bit are passed diagonally downwards instead of only to the right. Figure 4.7 shows a 4*4 carry save multiplier. An extra adder called vector merging adder is added to generate the final result. The resulting multiplier is called a carry save multiplier .In this the carry bits are not immediately added but are saved for the next adder stage. In the final stage ,carry and sum are merged using a carry lookahead adder. The worst case critical path is given as below by summing tadd=tcarry.

$t_{mult}=t_{and}+(N-1)t_{carry}+t_{merge}$



Vector Merging Adder

Fig.4.7: Carry Save Multiplier

Advantage
1.Shorter worst case critical path
Disadvantage
1.Increased area cost(because of the extra adder)

4.6.2.3. Tree Multiplier:

Partial sum adder rearranged in a tree like fashion to reduce both the critical path and number of adder cells. Consider four partial products each of which is four bits wide as shown in fig.4.8(a).Number of full adders needed for this operation is reduced by observing that only column 3 in the array has to add four bits. All other column are less complex as shown in fig 4.8(b).Now the original matrix of partial products is reorganized into tree shape to visually illustrate its varying depth. The challenge is to realize the complete matrix with a minimum depth and a minimum number of adder elements. The first type of operator used to cover the array is a full adder ,which takes three inputs and produces two outputs: the sum located in the same column and the carry located in the next one .For this reason, the full adder is called a 3-2 compressor. It is denoted by a circle covering three bits. The other operator is the half adder, which takes two input bits in a column and produces two outputs. The half adder is denoted by a circle covering two bits.

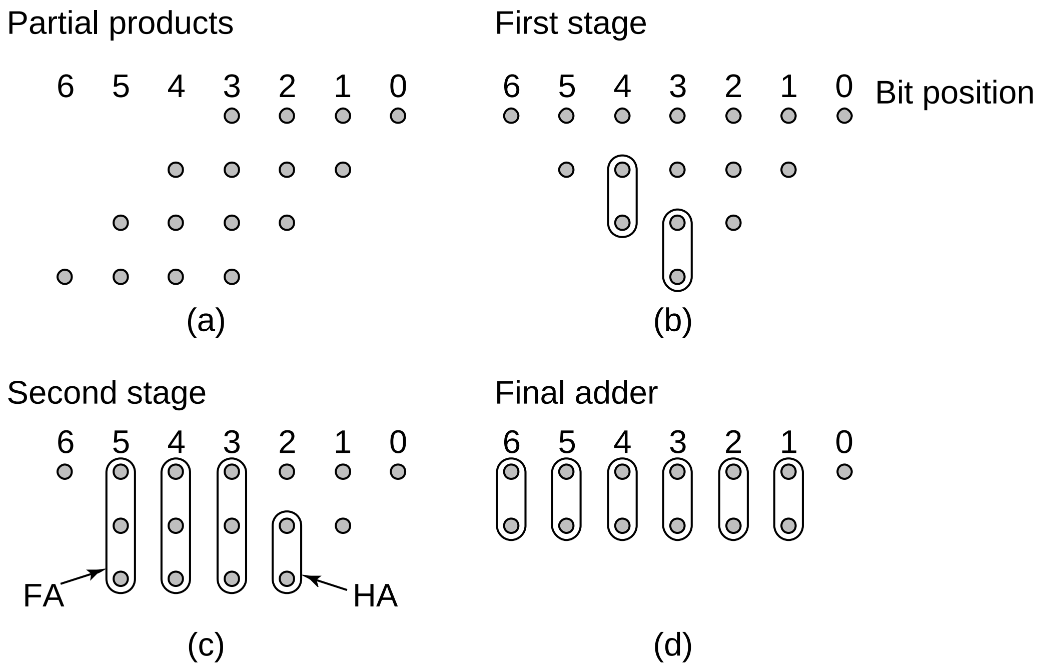


Fig.4.8: Tree Multiplier

To obtain minimal implementation, the tree is covered with full adders and half adders Starting from its densest part. First half adder is introduced in columns 4 and 3 as shown in figure 4.8(b).The reduced tree is shown in fig.4.8(c).A second round of reductions creates a tree of depth 2,shown in fig 4.8(d).Only three full adders and three half adders are used for the reduction process, compared with six full adders and six half adders in the carry save multiplier. The final stage consists of a simple 2 input adder, for which any type of adder can be used.

This structure is called the wallance tree multiplier and its implementation is shown in fig.4.9.The tree multiplier realizes substantial hardware savings for larger multipliers.

The propagation delay is reduced to O($\log_{3/2}$(N)).This structure is substantially faster than the carry save structure for large multiplier word lengths.

Disadvantages:
1.Very irregular
2.Complicated layout

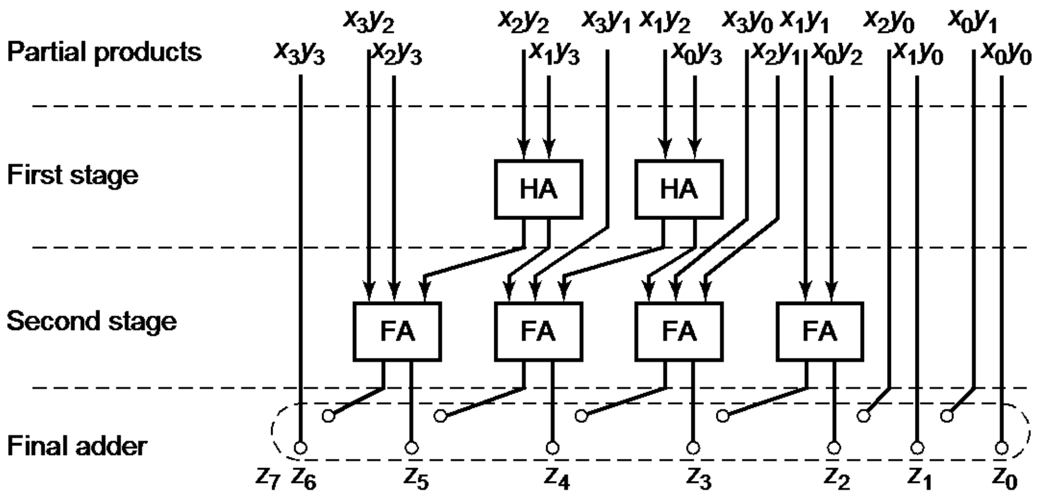


Fig.4.9: 4-bit wallance tree multiplier

4.6.3. Final Addition**:**

The final step for completing the multiplication is to combine the result in the final adder. The choice of the adder style depends on the structure of the accumulation array. A carry lookahead adder is the preferable option if all inputs bits to the adder arrive at the same time, as it yields the smallest possible delay.

4.7. Dividers:

4.7.1 Combinational Divider:

Let consider two unsigned binary numbers: D (m-bit dividend), and d (n-bit divisor). In this case, the division consists on finding two unsigned binary numbers, C (quotient) and r (remainder), $r<d$, such that:

$$D= C. d + r$$

The division is defined for $d \neq 0$. Therefore, in what follows it is assumed that this condition is met, i.e., before dividing, it is checked if $d \neq 0$, and only in this affirmative case, the division is performed.

With the condition $r<d$, C and r are unique, and to calculate them, an immediate combinational solution might be thought obviously. To implement the division it is sufficed, for example, a ROM of m + n address bits and m outputs (2^{m+n} words of m bits), in which the quotient and the remainder corresponding to every possible (D, d) are written. For real cases this is not a feasible combinational solution since m +n will result almost always too large to directly synthesize the corresponding functions (for example, to include all possible outputs in a ROM).

Another combinational solution is possible that attempts to mimic the division algorithm as a series of subtractions and shifts. Before addressing this alternative, more aspects about the operands have to be established. Specifically it is assumed that, as usual, the relationship between the lengths of the dividend and the divisor is m = 2n - 1, and that the most significant bit of the divisor, d, is 1. Neither of these assumptions implies restriction, because on the one hand, the size of the operand can always be adjusted by adding zeros, and, second, by shifting the divisor it is possible to make 1 the most significant bit; after division, the shifts made in the divisor must be properly transferred to the quotient and the remainder to obtain correct results. With these assumptions, with n-bits for both the quotient and to the remainder, all possible results may be represented, and the division is made in n-steps, in each of which a bit of the quotient is obtained.

In what follows, an example will be used to reach a combinational divider circuit: let n = 4, D = 0110101, d = 1011. The four stages of calculation for this case are detailed in Fig. 4.10. In the first stage d is subtracted from the four most significant bits of D (D6D5D4D3); if the result is positive (and therefore no output borrow), the quotient bit is 1, and the difference D6D5D4D3 - d passes to the next stage as the most significant bits of the modified dividend. If the result is negative (i.e., there is output borrow), the quotient bit is 0, and the dividend unchanged passes to the next stage. In other words, the quotient bit is the complement of the borrow of the subtractor output, and if the quotient bit is 0, D without changing is selected for the next stage, while if the quotient bit is 1, the most significant bits of the dividend bit must be modified selecting D6D5D4D3 - d. Therefore, with a full subtractor, FS, to take into account the possible borrow of the previous bit, plus one 2-to-1 multiplexer, the circuit necessary for processing each bit can be constructed, as shown with cell CR of Fig. 4.11. If for a given bit (as with the least significant bit) no input borrows are to be considered, the full subtractor FS can be replaced by a half subtractor, HS, resulting in the CS cell, simpler than the CR, Fig4.11.

The second and subsequent iterations consist on repeating the same as the first iteration, using in each case the unmodified or modified dividend which has resulted in the previous iteration. Then, by subtracting, the divisor is shifted one position to the right in each iteration. The remainder, r3… r0, is obtained in the fourth iteration.

The circuit for dividing a number of seven bits by other of four bits is detailed in Fig. 4.12, in which 12 CR cells and 7 CS cells are used (or 19 CR cells, if only one single type of cells want to be used). As it has been already indicated, the divisor has to be adjusted to get that always the most significant bit is a 1, and after division, these movements have to be translated to the results. It is straightforward to extend these design divisors for any value of n.

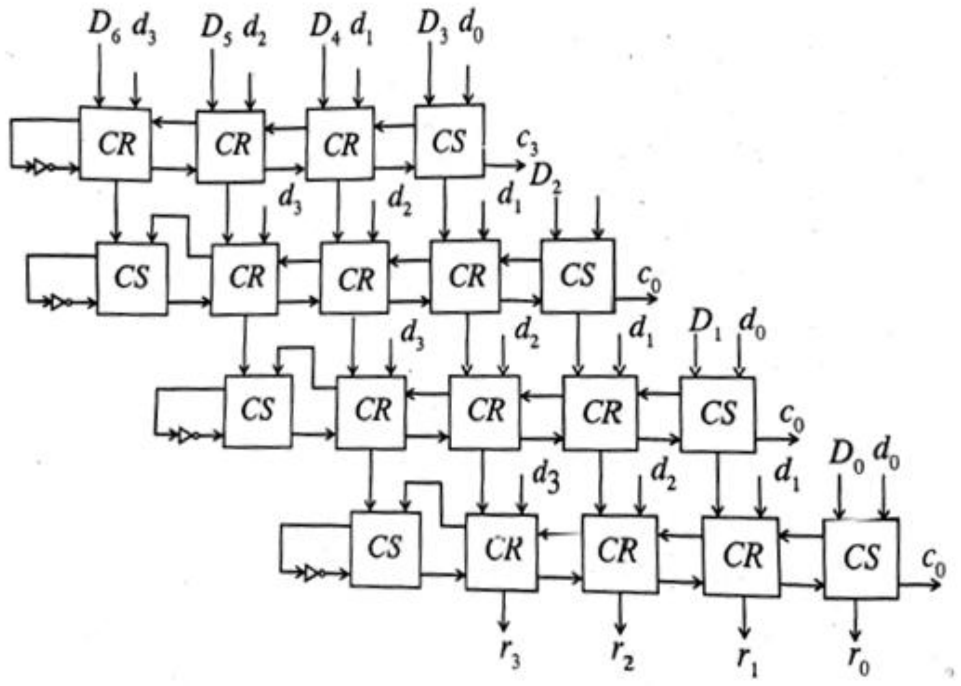


Fig.4.10: Combinational Divider

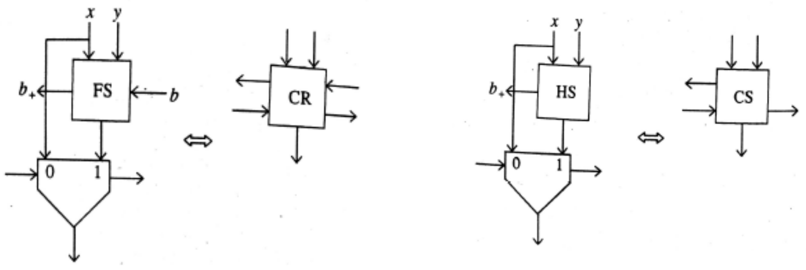


Fig.4.11:CR cell and CS cell

4.7.2 Sequential Divider**:**

The most common divisors are the sequential. The ideas that led to the divisor of Fig. 4.10. can be used to construct a divisor that divides D, of 2n - 1 bits, by d, of n bits, using n clock pulses. As a particular case it is still assumed that n = 4.

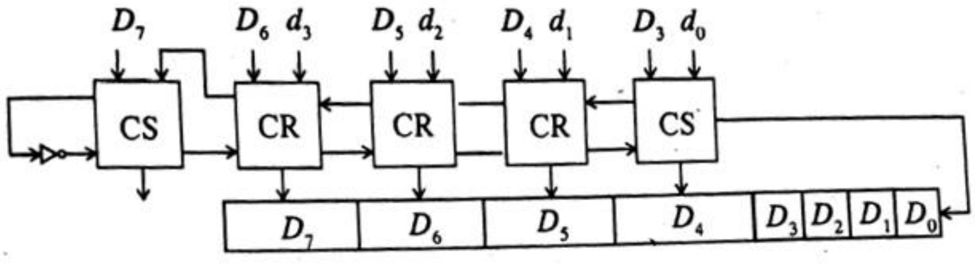


Fig.4.12: Sequential Divider:

Figure 4.11 shows a circuit using three CR cells, two CS cells, one 4-bit latch to store the divisor, d (this register it is not shown in Fig. 2.14e), and an 8-bit register for the dividend, D. This register D consists of two 4-bit register: the first register (D7D6D5D4) must be simultaneous reading and writing (i.e., master-slave), the second (D3D2D1D0) must be a shift register with serial input and output. The registers d and D have to be loaded with the data to be processed before starting operation. Obviously always D7 = 0 before starting to divide, and the divisor will have been shifted so the most significant bit of d is 1. The shift register (D3D2D1D0) is used to store the bits of the quotient. It is easily verified that this circuit in Fig. 4.11 does the same as the one in Fig. 4.10, unless using four clock pulses. Therefore, after four iterations, the quotient is stored in D3D2D1D0 and the remainder of the division is stored in D7D6D5D4. Again, this circuit can be extended immediately to any value of n.

4.7.3 Dividing by a Constant

In some applications, as in the scaling or in the change of base or in the modular reduction, it is necessary to divide a data set by the same constant. For this purpose different specific circuits can be used. In what follows it is assumed that integer data are going to be divided by an integer constant. It is easy to see that, strictly, just dividers to divide by odd numbers have to be designed. Indeed, the division of a number of m bits by 2n is simplified to n shifts: the n least significant bits are the remainder of the division, and the m - n most significant bits are the quotient. Moreover, any even number can be decomposed into the product of an odd number by a power of two:

$$C = I \times 2^n \Rightarrow \frac{N}{C} = \frac{N}{I}\frac{1}{2^n}$$

A first solution to design a divider by a constant, even or odd, consists of particularizing the generic circuits of 4.12 for the divisor to be used. For example, the sequential divisor of the Fig. 4.12 is shown in Fig. 4.13; this sequential divisor is particularized to divide by 10 (1010 = 10102) any unsigned 7-bit integer data. Obviously, the same result is obtained when dividing by 5, and then by 2. In any case, these particularized circuits provide both the quotient and the remainder of the division.

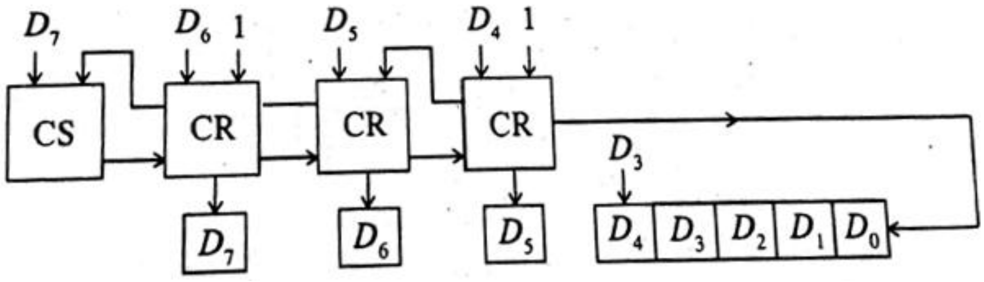


Fig.4.13: Dividing by a Constant

4.8.SHIFTERS:

Shift operation is an arithmetic operation. This requires adequate hardware support. Shifters are used in floating point units, scalers, and multiplications by constant numbers. Shifting a data word left or right over a constant amount is a trivial hardware operation. A programmable shifter is more complex and requires active circuitry. Such a shifter is an intrice multiplexer circuit. Figure 4.14 shows a simple one bit left right shifter. Depending on the control signals the input word is either shifted left or right, or unchanged. Multi-bit shifters are built by cascading a number of these units. This approach is complex, unwieldy, and too slow for larger shift values. The following are the two types of shifters.

1.Barrel Shifter

2.Logarithmic Shifter

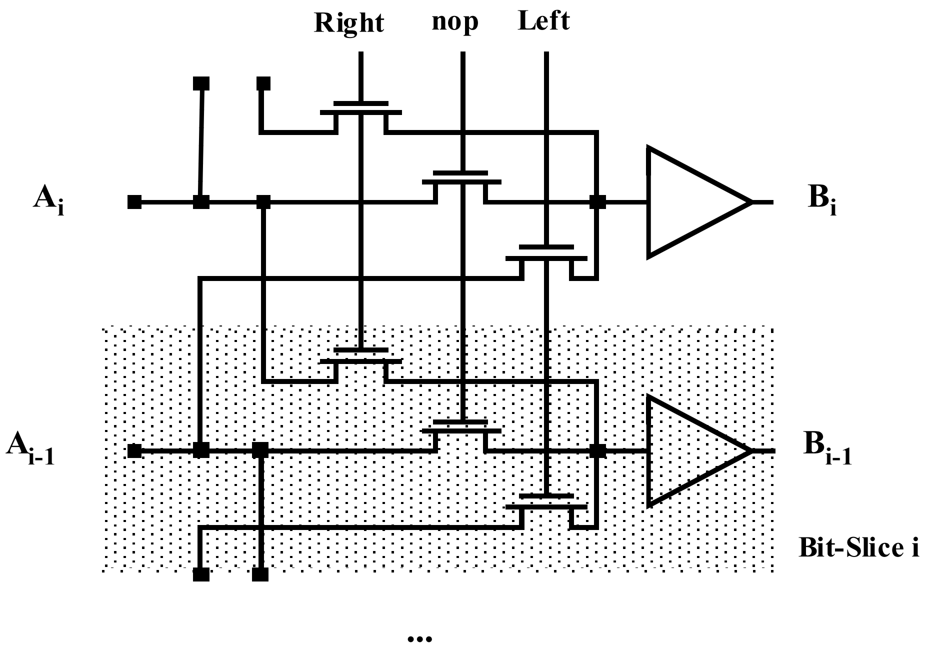


Fig.4.14: one bit left right shifter

4.8.1.Barrel Shifter:

Figure 4.15 shows the structure of a barrel shifter. It consists of an array of transistors ,in which the number of rows equals the word length of the data, and the number of columns equals the maximum shift width. In this case, both are equal to four. The control wires are routed diagonally through the array.

Advantages:

1. Signal has to pass through at most one transmission gate
2. Propagation delay is constant and independent of the shift value
3. Used for small shift values.

Disadvantages:

1. Capacitance at the input of the buffers rises linearly with the maximum shift width.

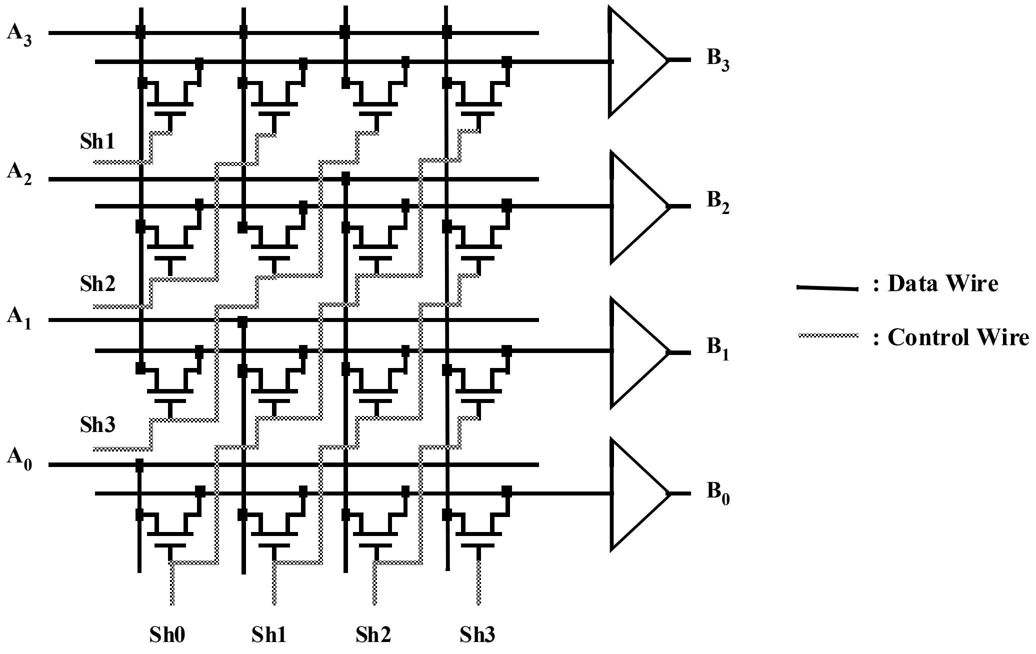


Fig.4.15: Barrel Shifter

Layout size in barrel shifter is not dominated by the active transistors, but by the number of wires running through the cell. The size of the cell is bounded by the pitch of the metal wires. When selecting a shifter, the format in which to present the shift value is provided. Barrel shifter needs a control wire for every shift bit. A 4-bit shifter needs four control signals .In a processor the required shift value comes in an encoded binary format. The encoded control word needs two control signal and is represented as 11 for a shift over three bits.

4.8.2.Logarithmic Shifter:

Logarithmic shifter uses staged approach for shifting. The total shift value is decomposed into shifts over powers of two. A shifter with a maximum shift width of M consists of a log2M stages, where the i th stage either shifts over 2i or passes the data unchanged. To shift over five bits, the first stage is set to shift mode, the second to pass mode, and the last stage again to shift. The control word for this shifter is already encoded and no separate decoder is required.

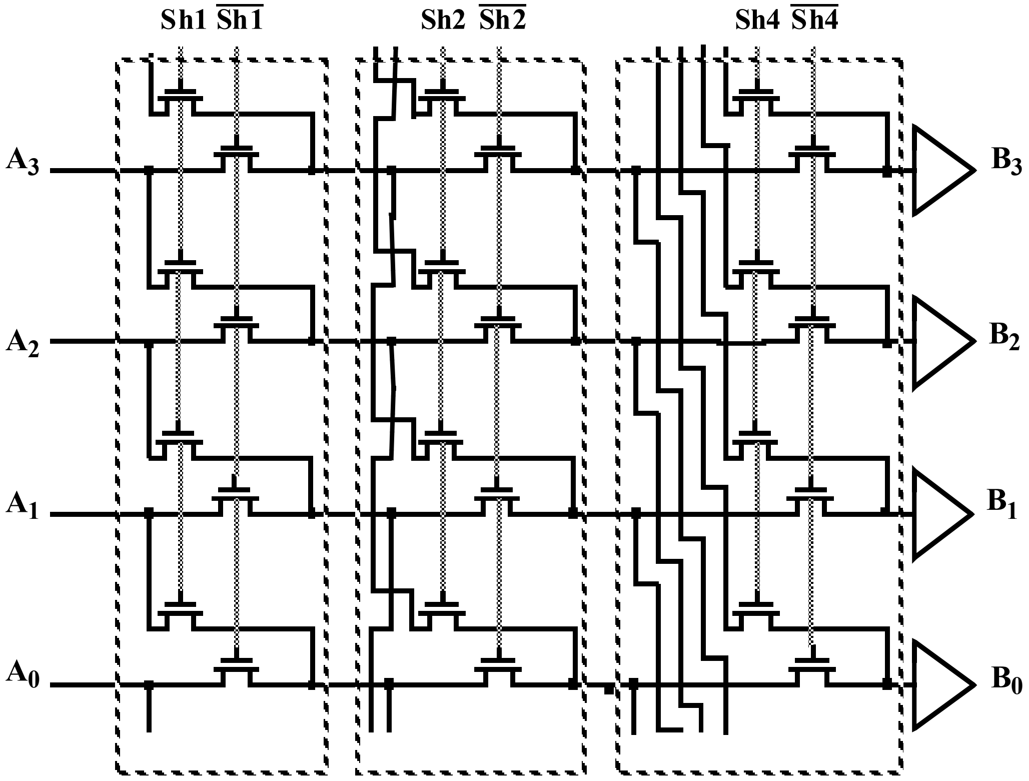

Sh1 $\overline{\text{Sh1}}$
Sh2 $\overline{\text{Sh2}}$
Sh4 $\overline{\text{Sh4}}$
A_3
A_2
A_1
A_0
B_3
B_2
B_1
B_0

Fig.4.16:Logarithmic shifter with maximal shift width of seven bits to the right

Speed of the logarithmic shifter depends on the shift width in a logarithmic way. An M-bit shifter requires log2M stages. The series connection of pass transistors slows down the shifter for larger shift vales. Also intermediate buffers are careful introduced. For larger shift values ,the logarithmic shifter becomes more effective ,in terms of both area and speed. The logarithmic shifter is easily parameterized, allowing for automatic generation.

4.9 Speed and area tradeoff:

With a fixed architecture of the datapath ,speed ,area ,and power can be traded off through the choice of the supply voltage, transistor threshold, and device sizes. This enables a variety of power minimization techniques, and is classified as follows.

1. Enable Time: Some designs are implemented or enabled at design time. Transistor width and length are fixed at the design time. Supply voltage and transistor thresholds are assigned statically during the design phase or changed dynamically at run time. Other techniques address

the time that a function or module is in idle or standby mode. The power dissipation of a module in sleep mode is absolutely minimal

2. Targeted Dissipation Sources:

This depends on the source of power dissipation like active (dynamic) power or leakage (static) power. Lowering supply voltage not only reduces the energy consumed per transition but also reduces the leakage current. A common method to reduce power in idle mode is the clock gating technique. Clock gating does not reduce the leakage power of the idle block. Complicated schemes to lower the standby power are used ,such as increasing the transistor thresholds or switching off the power rails. The following are the techniques to reduce the power.

1. Design Time Power Reduction
2. Run Time Power Management
3. Power Reduction is standby or sleep Mode
4 Design as a Tradeoff

<u>4.9.1.Design Time Power Reduction:</u>

The following are the techniques to reduce the power during design time

1. Supply voltage reduction
2. Using multiple supply voltage
3. Using multiple device thresholds
4. Reducing switching capacitance through transistor sizing
5. Reducing switching

<u>4.9.1.1 Supply voltage reduction</u>:

Supply voltage reducing results in quadratic power saving. But delay of CMOS gates increases with supply voltage. At datapath level, this loss of performance is compensated by logical and architectural optimizations. For example, a ripple carry adder is replaced by lookahead adder. Lookahead adder implementation is larger and more complex ,but run at a lower supply voltage for the same performance.

<u>4.11.1.2.Using multiple supply voltages</u>:

This technique selectively decrease the supply voltage on some of the gates which,

1. Correspond to fast path and finish the computation early
2. Drive larger capacitances, with increased delay

A separate supply voltage is provided for the I/O for compatibility. The logic core is powered from lower voltage supplies .Multiple voltages are assigned on a gate-by-gate basis.

The two different ways for using multiple supply voltages are,

1. Module level Voltage selection

2. Multiple supplies inside a block

Multiple supplies inside a block:

In this method ,individually set the supply voltage for each cell inside a block. Histogram of the critical path delays shows that only a few paths are critical or near critical path and that many paths have much shorter delays. The shorter paths waste energy, as there is no reward for finishing early. For each of these paths, the supply voltage is lowered to the optimum level. This is not easily achievable, as many logic gates are shared between different paths and it is impractical to generate and distribute a wide range as supply voltage. Figure 4.17 shows the implementation of employing two supply voltages using clustered voltage scaling technique, each path starts with the high supply voltage and switches to low supply when delay slack is available. Level conversion is performed in the registers at the end of the paths, using circuits shown in fig 4.18.

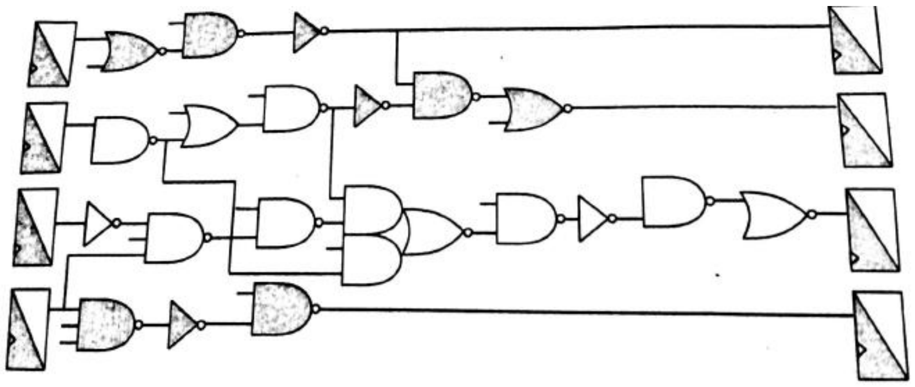


Fig.4.17: Dual power supply design

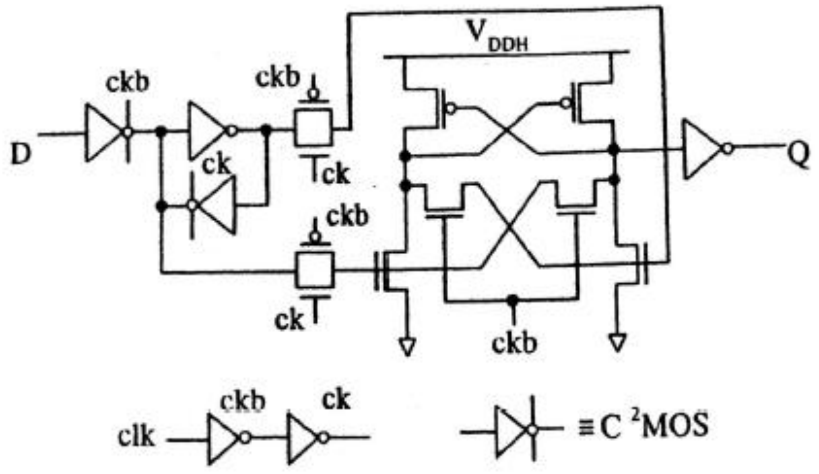


Fig.4.18: Level conversion register

Figure 4.19 shows the impact of dual supply design approach, which plots the path length distribution of a logic block for single and dual supplies. When a single supply is used, a large fraction of the path is substantially shorter than critical. Adding the second supply shifts the delay distribution closer to the target delay, distribution closer to the target delay ,making many more paths critical. If the design has very few critical paths and most of paths finish early ,energy savings would be large. On the other hand ,if most of the paths inside a block are critical introducing the second supply does not yield and energy savings.

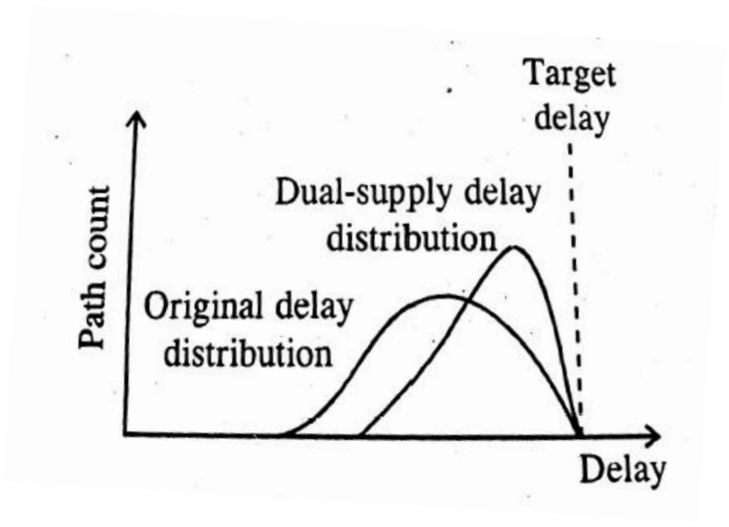


Fig.4.19: Path delay distribution

<u>4.9.1.3. Using Multiple Device Thresholds</u>:

The use of device with multiple thresholds offers another way of trading off speed for power. Most sub 0.25µm CMOS technologies offer two types of n-type and p-type transistors, with thresholds differing by about 100mv.Low threshold devices are used in timing critical paths, while the high thresholds are used anywhere else. The use of multiple thresholds does not require level converters or any other special circuits, as shown in fig.4.20.The assignment of thresholds can reduce the leakage power by 80%-90 %.

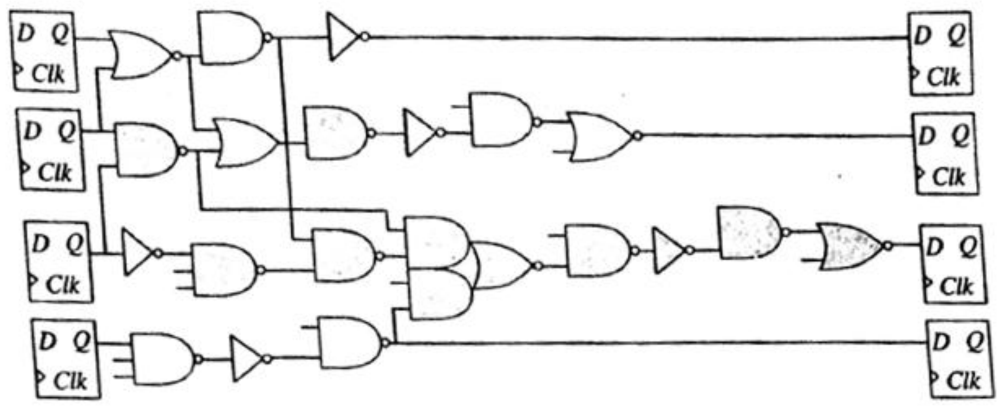


Fig.4.20: Dual threshold gates

<u>4.9.1.4. Reducing switching capacitance through transistor sizing</u>

Input capacitance of a complementary CMOS gate is proportion to its size and speed. The optimal sizing of gates in a logic path should be sized to have an effective fan-out of approximately 4 to get the minimum delay for that path. For an inverter chain with a given load and number of stages , the minimum delay is achieved when the size of each inverter in the chain is the geometric mean of its neighbors. When the minimum delay achieved by sizing is below the desired delay, an optimization problem is formulated that minimizes the switching capacitance under delay constrains. The optimal approach is to adjust the tapering factor at each stage.

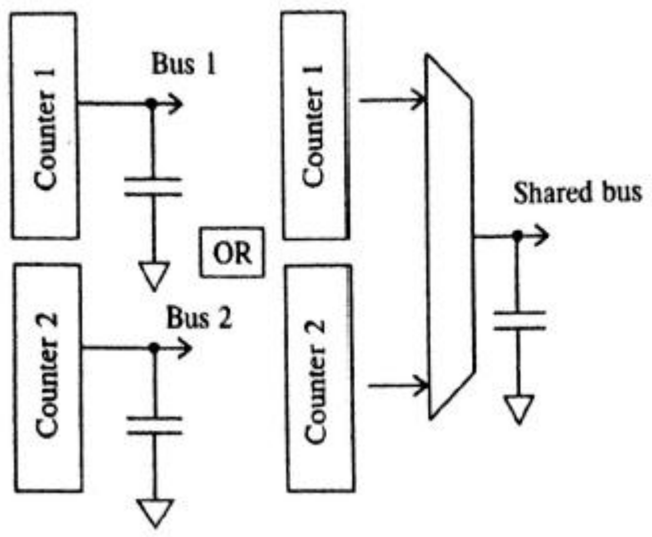
<u>4.9.1.5.Reducing Switching capacitance through logic and architecture optimization</u>:

Effective capacitance is the product of the physical capacitance and the switching activity. Logic and architectural optimizations reduce the switching activity without degrading the performance. The following are the methods of reducing switching activity.

1. Switching activity reduction by resource allocation
2. Glitch Reduction through path balancing

<u>Switching activity reduction by resource allocation:</u>

Multiplexing multiple operations on a single hardware unit has harmful effect on the power consumption. Fig 4.21 shows a simple experiment which compares the power consumption of two counters running simultaneously. In the first case both counters run on separate hardware .In the second case, both counters are multiplexed on the same unit. Fig 4.21 plots the number of switching events as a function of the skew between the two counters. The non multiplexed counter is always superior, except when both run in a completely synchronous fashion. The multiplexed counter tends to randomize the data signals presented to the operational unit, resulting in increased switching activity. Avoid excessive reuse of resources when power consumption is a concern. CMOS hardware units consume only negligible amounts of power when idle. Specialized operators only present an extra cost in area, while providing benefits in terms of speed and power.



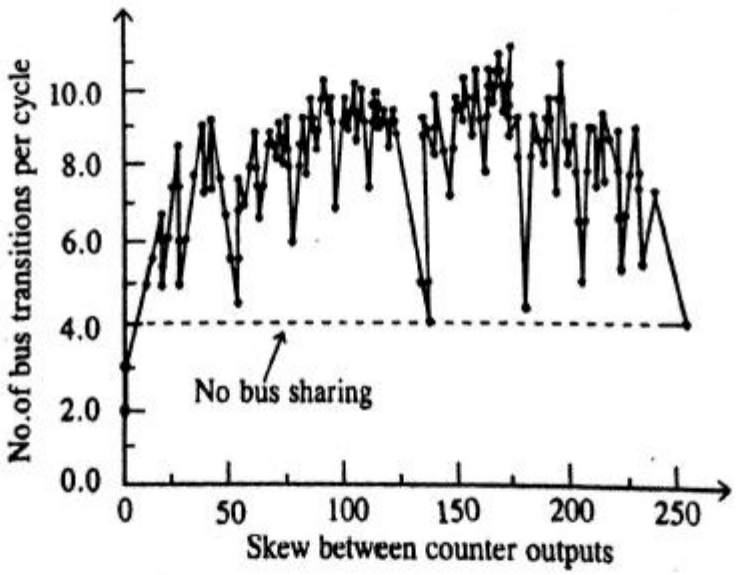


Fig.4.21: Switching Events

Glitch Reduction through path balancing:

The major power dissipation in adders and multipliers are due to dynamic hazards , or glitching. The transients in the length of the signal path are a cause. Fig 4.22 shows a simulated response of a 16-bit ripple adder for all inputs going simultaneously from 0 to 1.A number of the sum bits are shown as a function of time. The sum signals should be zero for all bits, but a 1 appears briefly at all of the outputs. The carry takes a significant amount of time to propagate from the first bit to the last.

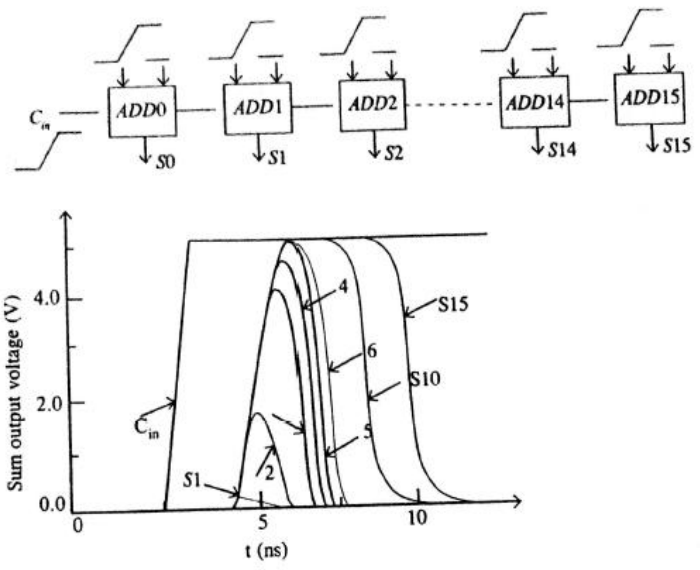


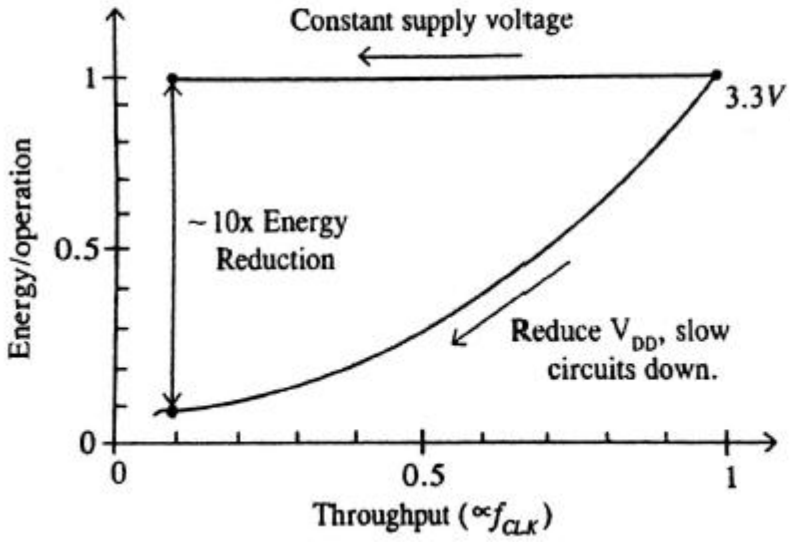
Fig.4.22: Glitching in 16-bit ripple carry Adder

4.9.2. Run Time power management:

The following methods are used for run time power management
1 .Dynamic supply voltage scaling (DVS)
2. Dynamic threshold scaling(DTS)

4.9.2.1.Dynamic supply voltage scaling (DVS):

Lowering clock frequency at reduced workloads reduces the power. But this does not save energy, because every operation is executed at high voltage level. If both supply voltage and frequency are lowered simultaneously, the energy is reduced. To maintain the required throughput for high workloads and minimize energy for low workloads, both supply and frequency must be dynamically varied according to the requirements application that is currently being executed. This technique is called dynamic voltage scaling (DVS).Fig 4.23 shows the analysis of energy/operation versus throughput for constant and variable supply voltage. It operates under the guidelines that a function should always be operated at the lowest supply voltage that means the timing constrains.



The DVS concept is enabled by the observation that the delay of most CMOS circuits and functions track each other well over a range of supply voltages, which is a necessity for system operation under varying supply conditions .Implementation of a DVS system consists of the following components:

1. Processor capable to operate under a wide variety of supply voltages
2. Supply regulation loop to set the minimum voltage necessary for operation at desire frequency
3. Operating system to calculate the frequency needed to get required throughput and complete the before deadline

Fig 4.24 shows a DVS system ia s ring oscillator, whose oscillating frequency matches the microprocessor critical path. Inside the power supply control loop, ring oscillator provides the translation between the supply voltage and the clock frequency. The operating system digitally sets the desired frequency(F_{DES}).The current value of ring oscillator frequency is measured and compared with the desired frequency .The difference is used as a feedback error (F_{ERR}).By adjusting the supply voltage ,the supply voltage loop changes the ring oscillator frequency to set the error value to 0.

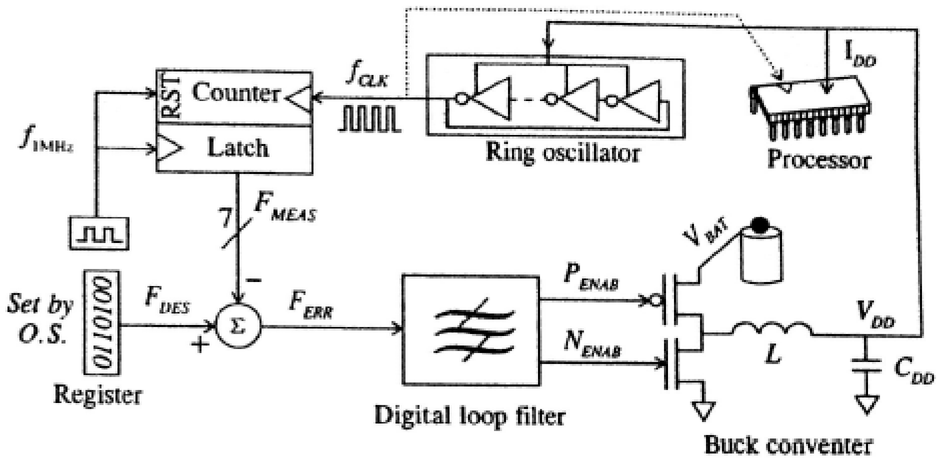


Fig.4.24:Block diagram of DVS using ring oscillator

4.9.2.2. Dynamic Threshold scaling:

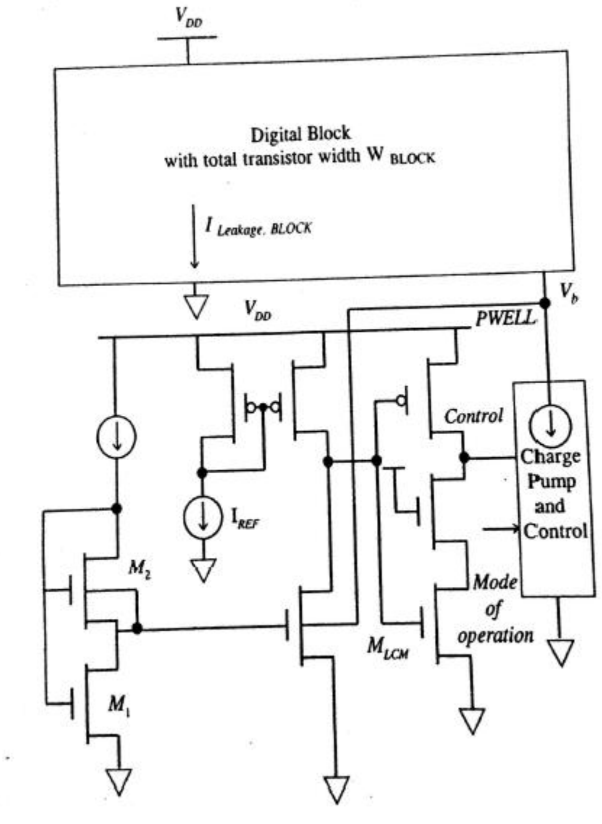


Fig.4.26:Variable threshold control scheme

For low latency computation, the threshold is lowered to its minimal value. For low speed computation it is increased .In the standby mode. it is set to the highest possible value to minimize the leakage current. Substract bias is the control to vary the threshold voltages dynamically. For this operate the transistors with four terminal in both n-type and p-type devices. Substrate biasing is implemented for a complete chip, on a block-by –block or a cell by cell basis.

Implementation of a DTS system is based on the feedback loop and has the following goals;

1. Lower leakage in standby mode
2. Compensate for threshold variations across the chip during normal operation
3. Control throughput to lower both active and leakage power based on performance requirements.

Current flow into the substrate is smaller than into the supply lines. Thus DTS has a smaller circuit overhead than DVS. Fig 4.26 shows a feedback system, designed to control the leakage in a digital module. This has a leakage control monitor and a substrate bias charge pump, added to the digital block of interest. Transistors M1 and M2 are biased to operate in the sub threshold region. When an nMOS transistor is in the sub-threshold ,its current is given by,

$$I_D = I_S/W_{0.}W.10^{\frac{V_{GS}}{S}}$$

The bias generator output, V_b is ,V_b=S.log(w_2/w_1)

<u>4.11.3.Power Reduction is standby or sleep mode</u>:

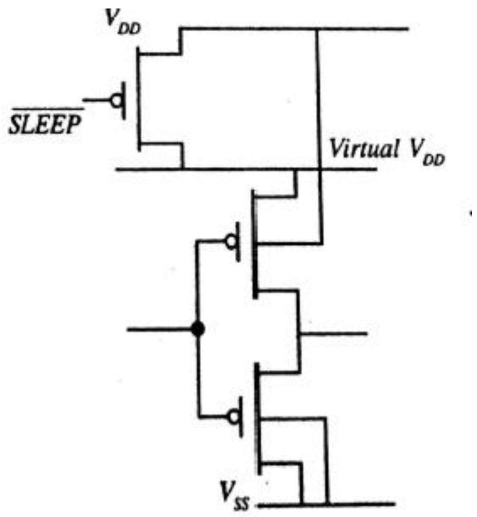


Fig.4.27:Sleep transistor used on supply rail

In idle mode no active switching occurs, all power dissipation is due to leakage.DTS technique is used to reduce the leakage during standby. A simple power down scheme utilizes large sleep transistors to switch OFF the power supply rails when the circuit is in the sleep mode. This approach reduces the leakage current, but increases the design complexity. This is implemented by using a power switch on the supply rail as shown in fig.4.27 or on both supply and ground rails as shown in fig 4.28.

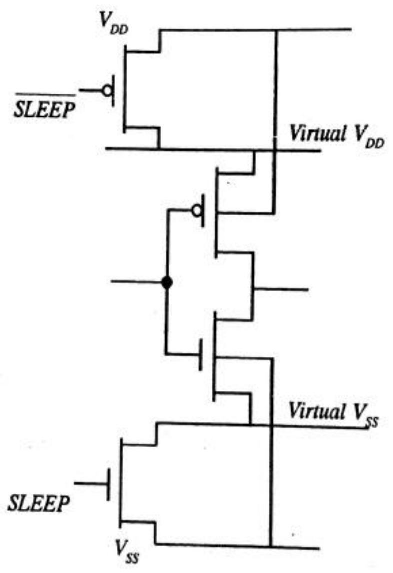


Fig.4.28: Sleep transistor used on both supply and ground rails

The sleep signal is high in normal operation, and the sleep transistors is small as a resistance as possible .Transistors finite resistance results in noise on supply rails. Sizing and threshold selection of the sleep transistors is subject to a trade off process. To minimize fluctuations in the supply voltage , the sleep transistors have a very low on resistance, and are very wide. With high threshold transistors leakage is reduced. Addition of sleep transistor increases the transistor stack height. This results in leakage reductions in the order of ten thousand times for low/high threshold switches.

4.9.4.Design as a tradeoff:

The analysis of the adder and multiplier circuits makes it clear that digital circuits design is a tradeoff between area, speed and power requirements. Fig 4.29 shows performance analysis of propagation delay as a function of the number of bits N. Fig 4.30 shows performance analysis of area as a function of the number of bits N.

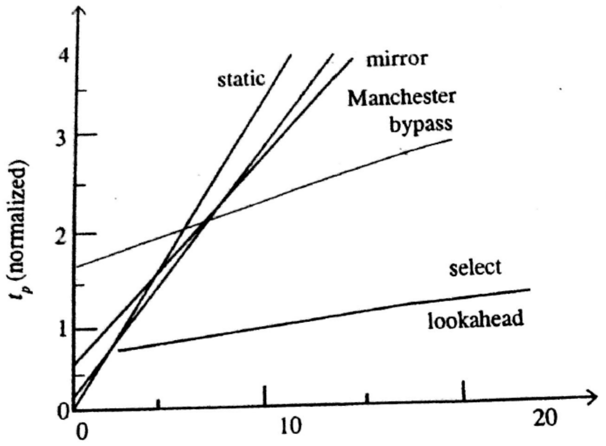


Fig.4.29: Performance analysis of various adder for propagation delay vs number of bits

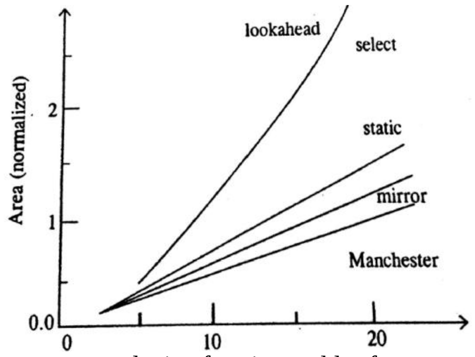


Fig.4.30: Performance analysis of various adder for area vs number of bits

The die area has a strong impact on the cost of an integrated circuit. A larger chip size means that fewer parts fit on a single wafer. Reducing the area decides the viability of a product .All design constraints like speed, power, and area contribute to the feasibility or market success of a design. The important design concepts are.,

1. Select the right structure before starting an elaborate circuit optimization. Optimizing transistor sizes and topologies probably will not give you the best result.
2. Determines the critical timing path through the circuit, and focus optimization efforts on that part of the circuit. Computer aided design tools are available to find the critical paths and size the transistors. Some noncritical paths are downsized to reduce power consumption
3. Circuit size is not only determined by the number and size of transistors ,but also by other factors such as wiring and the number of vias and contacts.
4. Optimization helps to get better result, but results in an irregular and convoluted topology.

5. Power and speed can be traded off through a choice of circuit sizing ,supply voltages and transistor thresholds.