Data Path Subsystems

4.1 Introduction

Most digital functions can be divided into the following categories:

1. Data path 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-of between speed, density, programmability, ease of design, and other variables. This chapter addresses design options for common data path operators, arrays, especially those used for memory. Control structures are most commonly coded in a hardware description language and synthesized.

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 ow in one direction, while control signals are introduced in a direction orthogonal to the data ow.

Common data path operators considered in this chapter include adders, one/zero detectors, comparators, counters, shifters, ALUs, and multipliers.

4.2 Shifters

Consider a direct MOS switch implementation of a 4X4 crossbar switch as shown in Fig. 4.1. The arrangement is quit general and may be readily expanded to accommodate n-bit inputs/outputs. In fact, this arrangement is an overkill in that any input line can be connected to any or all output lines-if all switches are closed, then all inputs are connected to all outputs in one glorious short circuit.

Furthermore, 16 control signals (sw_00)-sw_15, one for each transistor switch, must be provided to drive the crossbar switch, and such complexity is highly undesirable.

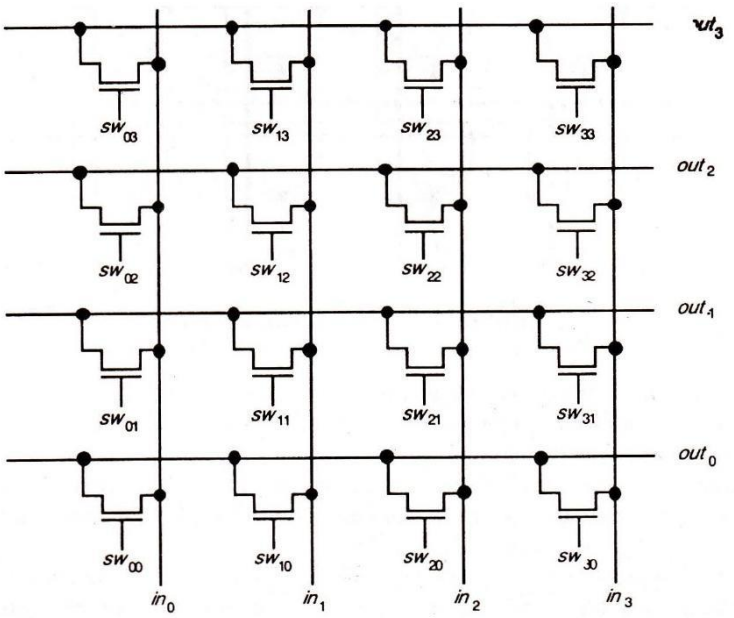


Figure 4.1: 4 x 4 crossbar switch.

An adaption of this arrangement) recognizes the fact that we can couple the switch gates together in groups of four (in this case) and also form four separate groups corresponding to shifts of zero, one, two, and three bits. The arrangement is readily adapted so that the in lines also run horizontally (to confirm the required strategy). The resulting arrangement is known as barrel shifter and a 4X4-bit barrel shifter circuit diagram is given in Fig. 4.2. The inter bus switches have their gate inputs connected in staircase fashion in group of four and there are now four shift control inputs which must be mutually exclusive in active state. CMOS transmission gates may be used in place of the simple pass transistor switches if appropriate.

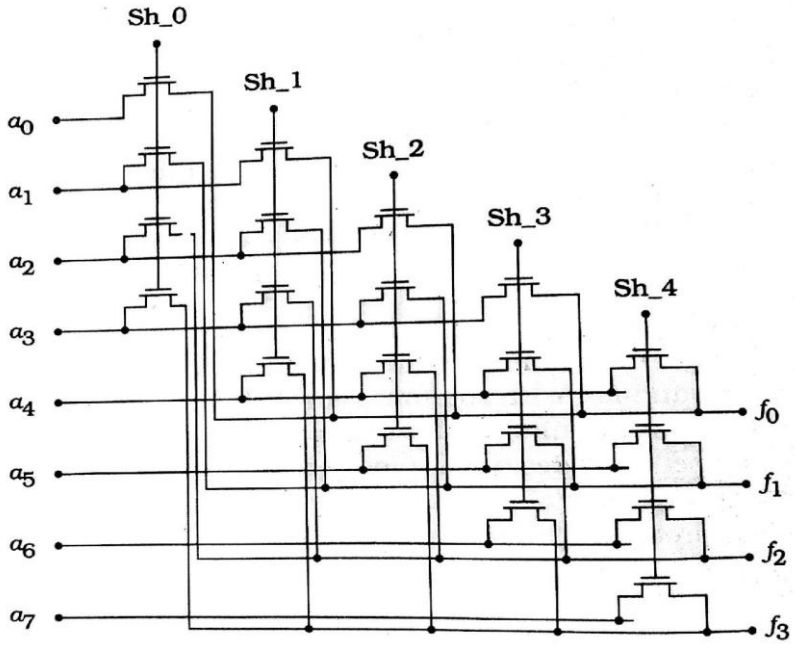


Figure 4.2: Barrel shifter

4.3 Adders

Addition is one of the basic operation perform in various processing like counting, multiplication and altering. Adders can be implemented in various forms to suit different speed and density requirements.

The truth table of a binary full adder is shown in Figure 4.3, along with some functions that will be of use during the discussion of adders. Adder inputs: A, B, Carry input

C	A	B	A.B (G)	A+B (P)	$A \oplus B$	SUM	CARRY
0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	0
0	1	0	0	1	1	1	0
0	1	1	1	1	0	0	1
1	0	0	0	0	0	1	0
1	0	1	0	1	1	0	1
1	1	0	0	1	1	0	1
1	1	1	1	1	0	1	1

Figure 4.3: Full adder truth table

Output: SUM, Carry output: CARRY Generate signal: G (A B); occurs when CARRY is internally generated within adder.
Propagate signal: P (A + B); when it is 1, C is passed to CARRY.
In some adders A, B is used as the P term because it may be reused to generate the sum term.

4.3.1 Single-Bit Adders

Probably the simplest approach to designing an adder is to implement gates to yield the required majority logic functions.

From the truth table these are:

The direct implementation of the above equations is shown in Fig. 4.4 using the gate schematic and the transistors is shown in Fig. 4.5.

$$S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$
$$S = C(AB + \overline{A}\,\overline{B}) + \overline{C}(\overline{A}B + A\overline{B})$$
$$S = C\overline{(A \oplus B)} + \overline{C}(A \oplus B)$$
$$S = C \oplus (A \oplus B)$$

$$C(i+1) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$
$$C(i+1) = AB(\overline{C} + C) + C(\overline{A}B + A\overline{B})$$
$$C(i+1) = AB + C(A \oplus B)$$

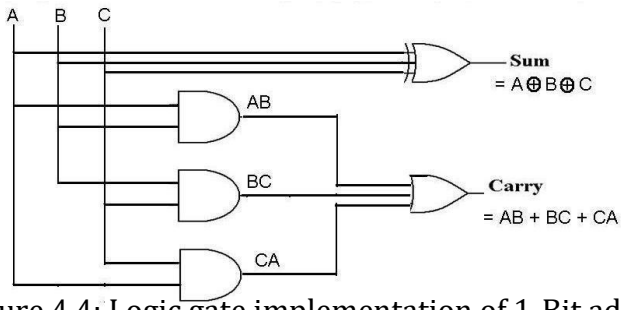


Figure 4.4: Logic gate implementation of 1-Bit adder

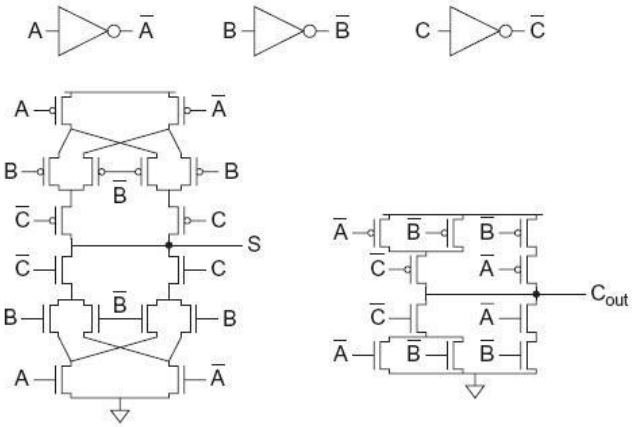


Figure 4.5: Transistor implementation of 1-Bit adder

The full adder of Fig. 4.5 employs 32 transistors (6 for the inverters, 10 for the carry circuit, and 16 for the 3-input XOR). A more compact design is based on the observation that S can be factored to reuse the CARRY term as follows:

For the **SUM** (S) SUM = (A XOR B) XOR Cin = (A ⊕ B) ⊕ Cin

For the **CARRY-OUT** (Cout) bit CARRY-OUT = A AND B OR Cin (A XOR B) = A.B + Cin (A ⊕ B)

Such a design is shown at the transistor levels in Figure 4.6 and uses only 28 transistors. Note that the pMOS network is complement to the nMOS network.

Here Cin=C

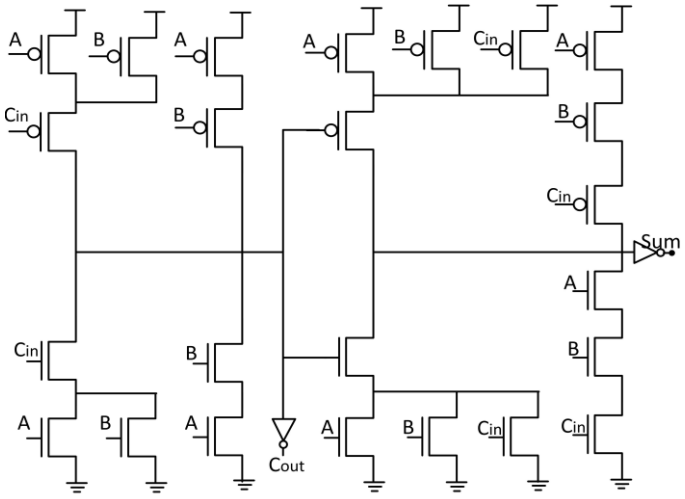


Figure 4.6: Transistor implementation of 1-Bit adder

4.3.2 n-Bit Parallel Adder or Ripple Carry Adder

A ripple carry adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascaded, with the carry output from each full adder connected to the carry input of the next full adder in the chain. Figure 4.7 shows the interconnection of four full adder (FA) circuits to provide a 4-bit ripple carry adder. Notice from Figure 4.7 that the input is from the right side because the reset cell traditionally represents the least significant bit (LSB). Bits a_0 and b_0 in the figure represent the least significant bits of the numbers to be added. The sum output is represented by the bits S_0-S_3.

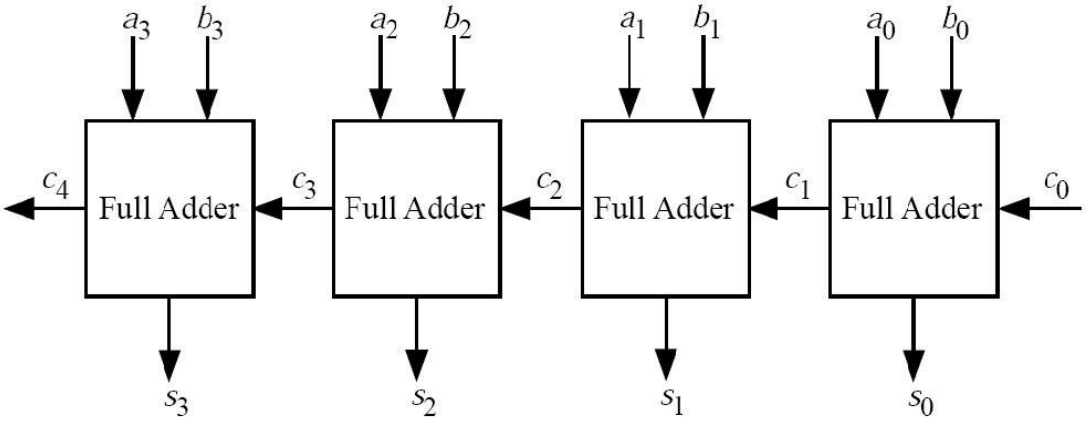


Figure 4.7: 4-bit ripple carry adder

The worst-case delay of the RCA is when a carry signal transition ripples through all stages of adder chain from the least significant bit to the most significant bit, which is approximated by:

$$t = (n-1)t_c + t_s$$

Where tc is the delay through the carry stage of a full adder, and ts is the delay to compute the sum of the last stage. The delay of ripple carry adder is linearly proportional to n, the number of bits, therefore the performance of the RCA is limited when n grows bigger. The advantages of the RCA are lower power consumption as well as a compact layout giving smaller chip area.

4.3.3 Carry look ahead adder (CLA)

The carry look ahead adder (CLA) solves the carry delay problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases:

(1) When both bits a_i and b_i are 1, or

(2) When one of the two bits is 1 and the carry-in is 1 . Thus, one can write,

$$c_{i+1} = a_i.b_i + (a_i \oplus b_i).c_i$$

$$s_i = (a_i \oplus b_i) \oplus c_i$$

The above two equations can be written in terms of two new signals P_i and G_i, which are shown in Figure 4.8:

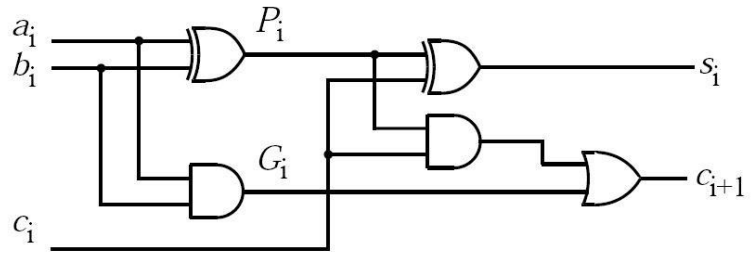


Figure 4.8: Full adder stage at i with P_i and G_i shown

$$c_{i+1} = G_i + P_i.c_i,\ s_i = P_i \oplus c_i, \quad \text{Where } G_i = a_i.b_i$$

P_i and G_i are called carry propagate and carry generate terms, respectively. Notice that the generate and propagate terms only depend on the input bits and thus will be valid after one and two gate delay, respectively. If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple.

Through all the previous stages to find its proper value. Let's apply this to a 4-bit adder to make it clear.

Putting i = 0; 1; 2; 3 in c_{i+1}

$c_1 = G_0 + P_0.c0$

$c_2 = G_1 + P_1.G_0 + P_1.P_0c_0$ $= G_i + P_i.c_i$ we get

$c_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.c_0$

$c_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3:.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.c_0$

Notice that the carry-out bit, c_{i+1}, of the last stage will be available after four delays: two gate delays to calculate the propagate signals and two delays as a result of the gates required to implement Equation c_4.

Figure 4.9 shows that a 4-bit CLA is built using gates to generate the P_i and $\mathbf{P_i = (a_i \oplus b_i)}$ G_i and signals and a logic block to generate the carry out signals according to Equations c_1 to c_4.

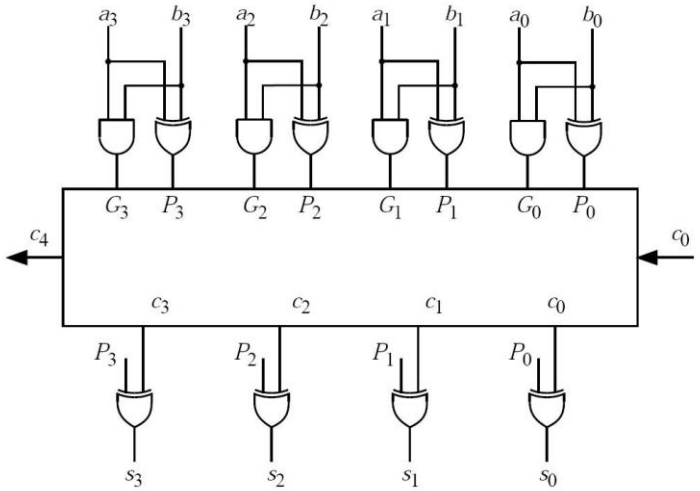
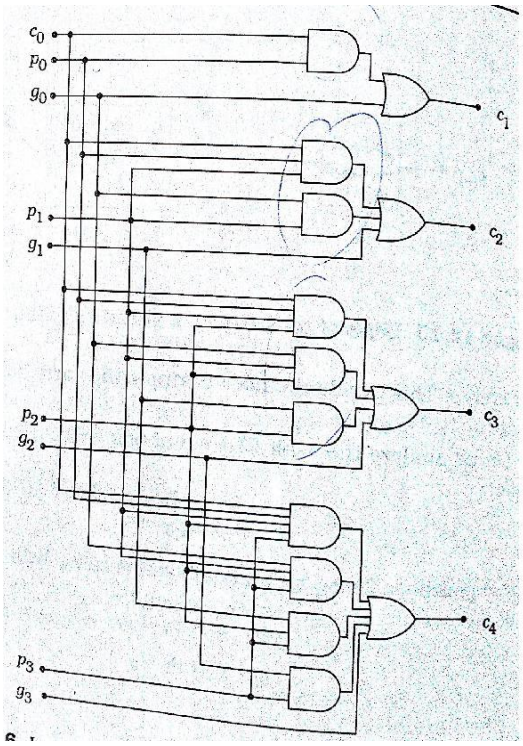


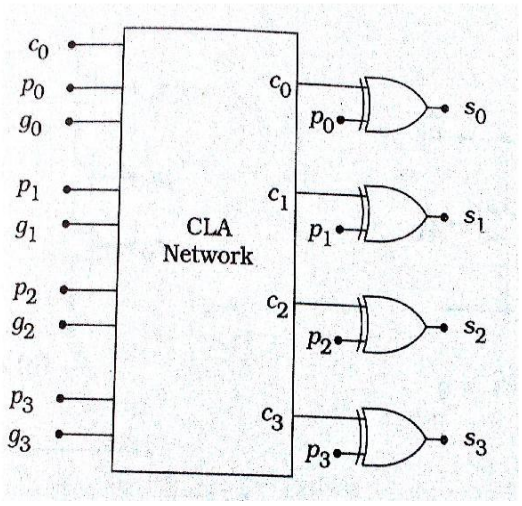
Figure 4.9: 4-Bit carry look ahead adder implementation in detail

Logic gate and transistor level implementation of carry bits are shown in Figure 4.10.

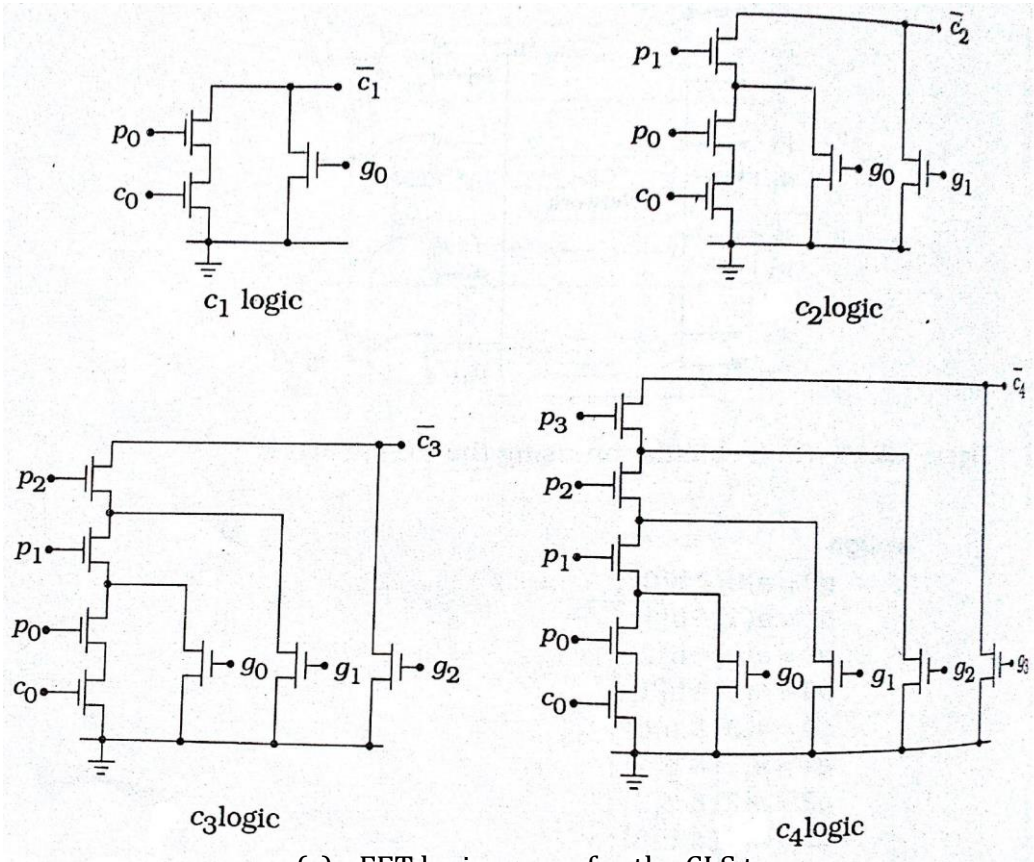
The disadvantage of CLA is that the carry logic block gets very complicated for more than 4-bits. For that reason, CLAs are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits.



(a) Logic network for 4-bit CLA carry bits



(b) Sum calculation using CLA network



(c) nFET logic arrays for the CLS terms

Figure 4.10: Carry structures of CLA

4.3.4 *Carry Skip Adder:*

As the name indicates, Carry Skip Adder (CSkA) uses skip logic in the propagation of carry . It is designed to speed up the addition operation by adding a propagation of carry bit around a portion of entire adder. The carry-in bit designated as C_i. The output of RCA (the last stage) is C_{i+4}. The Carry Skip circuitry consists of two logic gates. AND gate accepts the carry-in bit and compares it with the group of propagated signals.

$$\mathbf{P_i, P_{i+3}= (P_{i+3})*(P_{i+2})*(P_{i+1})*P_i \ \& \ Carry= C_{i+4} + (P_{i,\,i+3})* C_i.} \qquad \mathbf{(1)}$$

The architecture of CSkA is shown in Figure.

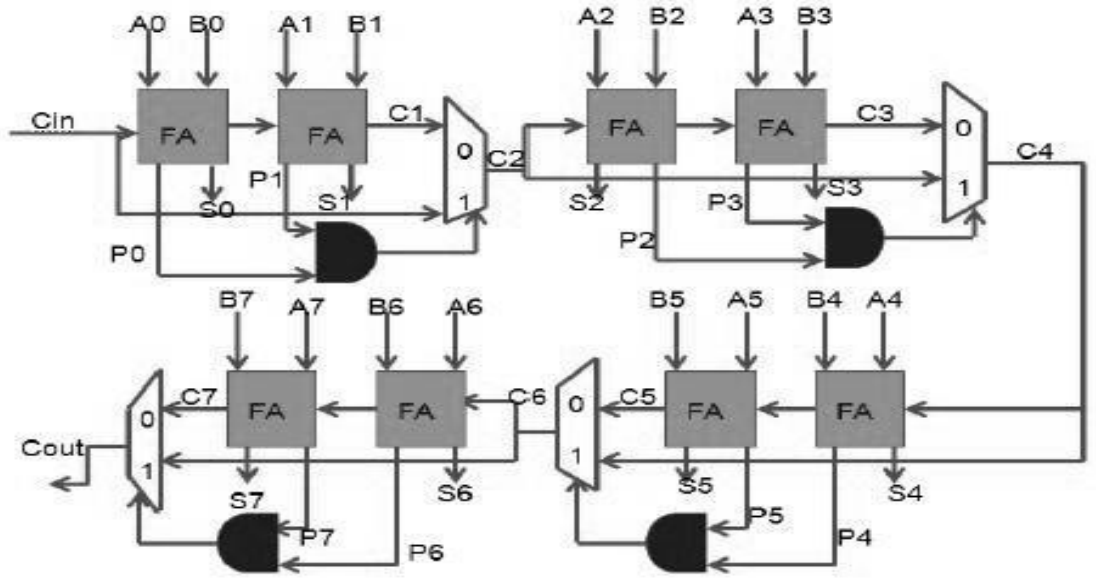


Fig. Carry Skip Adder (CSkA)

4.3.5 *Carry Save Adder:*

In Carry Save Adder (CSA), three bits are added parallelly at a time. In this scheme, the carry is not propagated through the stages. Instead, carry is stored in present stage, and updated as addend value in the next stage. Hence, the delay due to the carry is reduced in this scheme.

The architecture of CSA is shown in Fig.

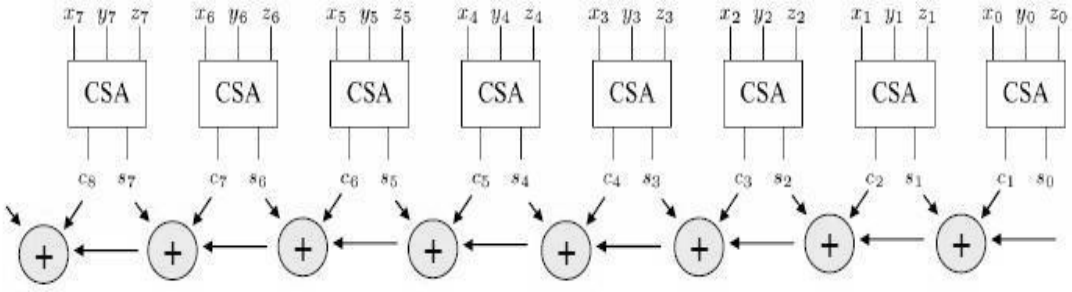


Fig. Carry save Adder (CSA)

4.3.6 *Carry Select Adder:*

Carry Select Adder (CS*I*A) architecture consists of independent generation of sum and carry i.e., C_{in}=1 and C_{in}=0 are executed parallelly [4]. Depending upon C_{in}, the external multiplexers select the carry to be propagated to next stage. Further, based on the carry input, the sum will be selected. Hence, the delay is reduced. However, the structure is increased due to the complexity of multiplexers [4].The architecture of CS*I*A is illustrated in Fig .

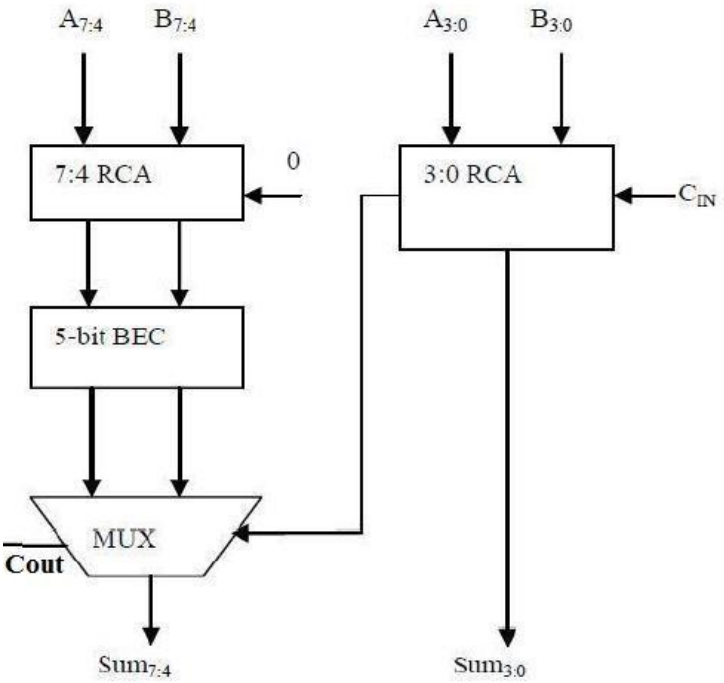


Fig. Carry Select Adder (CS*I*A)

4.3.7 *Carry Skip (Bypass) Adder:*

In Carry Bypass Adder (CBA), RCA is used to add 4-bits at a time and the carry generated will be propagated to next stage with help of multiplexer using select input as Bypass logic. By pass logic is formed from the product values as it is calculated in the CLA. Depending on the carry value and by pass logic, the carry is propagated to the next stage.

The architecture of Carry Bypass Adder (CBA) is given in Fig .

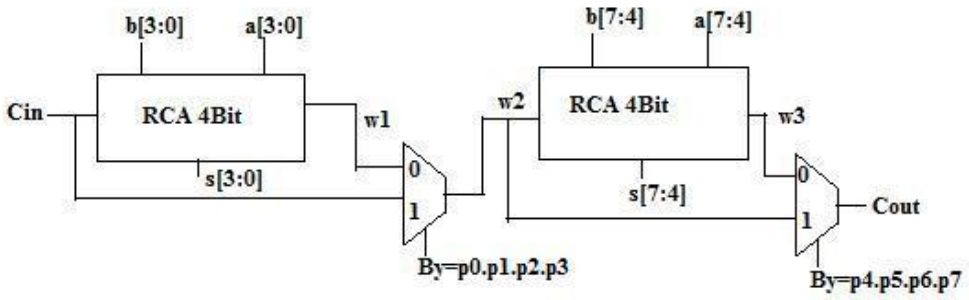


Fig. Carry Bypass Adder (CBA)

4.3.8 Manchester carry chain

This implementation can be very performant (20 transistors) depending on the way the XOR function is built. The carry propagation of the carry is controlled by the output of the XOR gate. The generation of the carry is directly made by the function at the bottom. When both input signals are 1, then the inverse output carry is 0. In the schematic of Figure 4.11, the carry passes through a complete transmission gate.

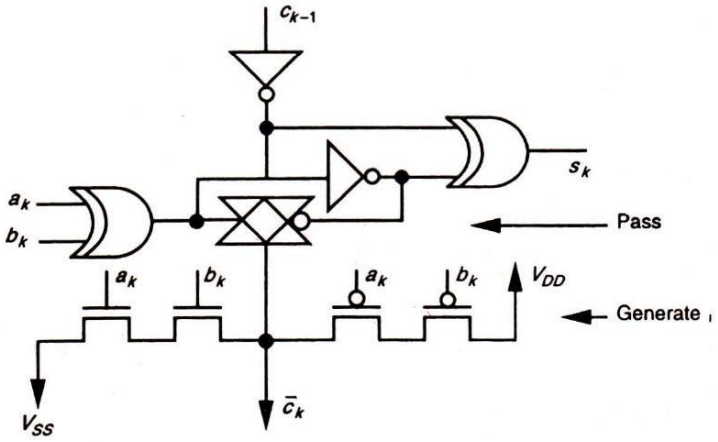


Figure 4.11: An adder element based on the pass/generate concept.

If the carry path is precharged to VDD, the transmission gate is then reduced to a simple NMOS transistor. In the same way the PMOS transistors of the carry generation is removed. One gets a Manchester cell.

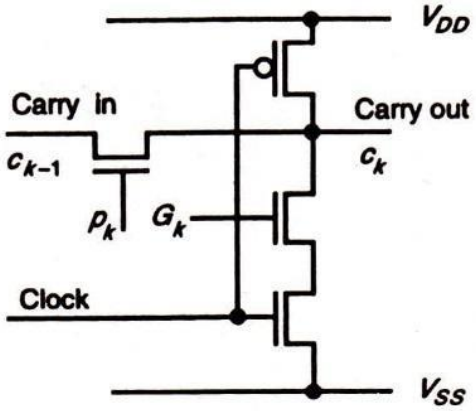


Figure 4.12: Manchester cell

The Manchester cell is very fast, but a large set of such cascaded cells would be slow. This is due to the distributed RC effect and the body effect making the propagation time grow with the square of the number of cells. Practically, an inverter is added every four cells, like in Figure 4.12.

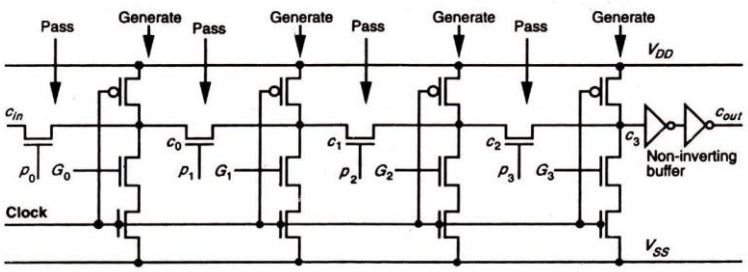


Figure 4.13: Cascaded Manchester carry-chain elements with buffering

4.4 Multipliers

In many digital signal processing operations - such as correlations, convolution, filtering, and frequency analysis - one needs to perform multiplication. The most basic form of multiplication consists of forming the product of two positive binary numbers. This may be accomplished through the traditional technique of successive additions and shifts, in which each addition is conditional on one of the multiplier bits. Here is an example.

multiplicand $1100 : 12_{10}$
multiplier $0101 : 5_{10}$

$$\begin{array}{r} 1100 \\ 0000 \\ 1100 \\ 0000 \\ \hline 0111100 \end{array} : 60_{10}$$

Figure 4.14: 4-bit multiplication

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

- Evaluation of partial products.
- Accumulation of the shifted partial products.

It should be noted that binary multiplication is equivalent to a logical AND operation. Thus evaluation of partial products consists of the logical ANDing of the multiplicand and the relevant multiplier bit. Each column of partial products must then be added and, if necessary, any carry values passed to the next column.

There are a number of techniques that may be used to perform multiplication. In general, the choice is based on 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:-

- Serial form
- Serial/parallel form
- Parallel form

4.4.1 Array Multiplication (Braun Array Multiplier)

A parallel multiplier is based on the observation that partial products in the multi-plication process may be independently computed in parallel. For example, consider the unsigned binary integers X and Y.

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

$$\begin{aligned} P = X \times Y &= \sum_{i=0}^{n-1} X_i 2^i \cdot \sum_{j=0}^{n-1} Y_j 2^j \\ &= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} (X_i Y_j) 2^{i+j} \\ &= \sum_{k=0}^{n+n-1} P_k 2^k \end{aligned}$$

Thus P_k are the partial product terms called summands. There are mn summands, which are produced in parallel by a set of mn AND gates.

For 4-bit numbers, the expression above may be expanded as in the table below.

				X3	X2	X1	X0	Multiplicand
				Y3	Y2	Y1	Y0	Multiplier
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 4.15 An nxn multiplier requires

$(n\ 1)^2$full adders, n 1 half adders, and n^2 AND gates.

The worst-case delay associated with such a multiplier is $(2n + l)t_g$, where t_g is the worst-case adder delay.

Cell shown in Figure 4.16 is a cell that may be used to construct a parallel multiplier.

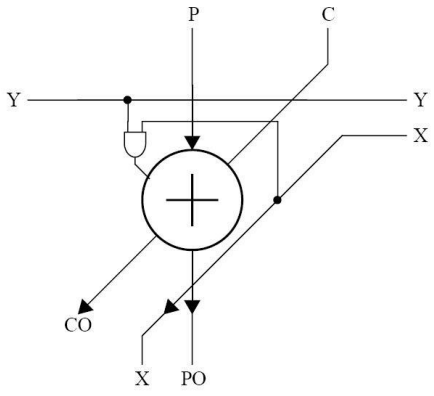


Figure 4.16: Basic cell to construct a parallel multiplier

The X_i term is propagated diagonally from top right to bottom left, while the y_j term is propagated horizontally. Incoming partial products enter at the top. Incoming CARRY IN values enter at the top right of the cell. The bit-wise AND is performed in the cell, and the SUM is passed to the next cell below. The CARRY 0UT is passed to the bottom left of the cell.

Figure 4.17 depicts the multiplier array with the partial products enumerated. The Multiplier can be drawn as a square array, as shown here, Figure 4.18 is the most convenient for implementation.

In this version the degeneration of the first two rows of the multiplier are shown. The first row of the multiplier adders has been replaced with AND gates while the second row employs half-adders rather than full adders.

This optimization might not be done if a completely regular multiplier were required (i.e. one array cell). In this case the appropriate inputs to the first and second row would be connected to ground, as shown in the previous slide. An adder with equal carry and sum propagation times is advantageous, because the worst-case multiply time depends on both paths.

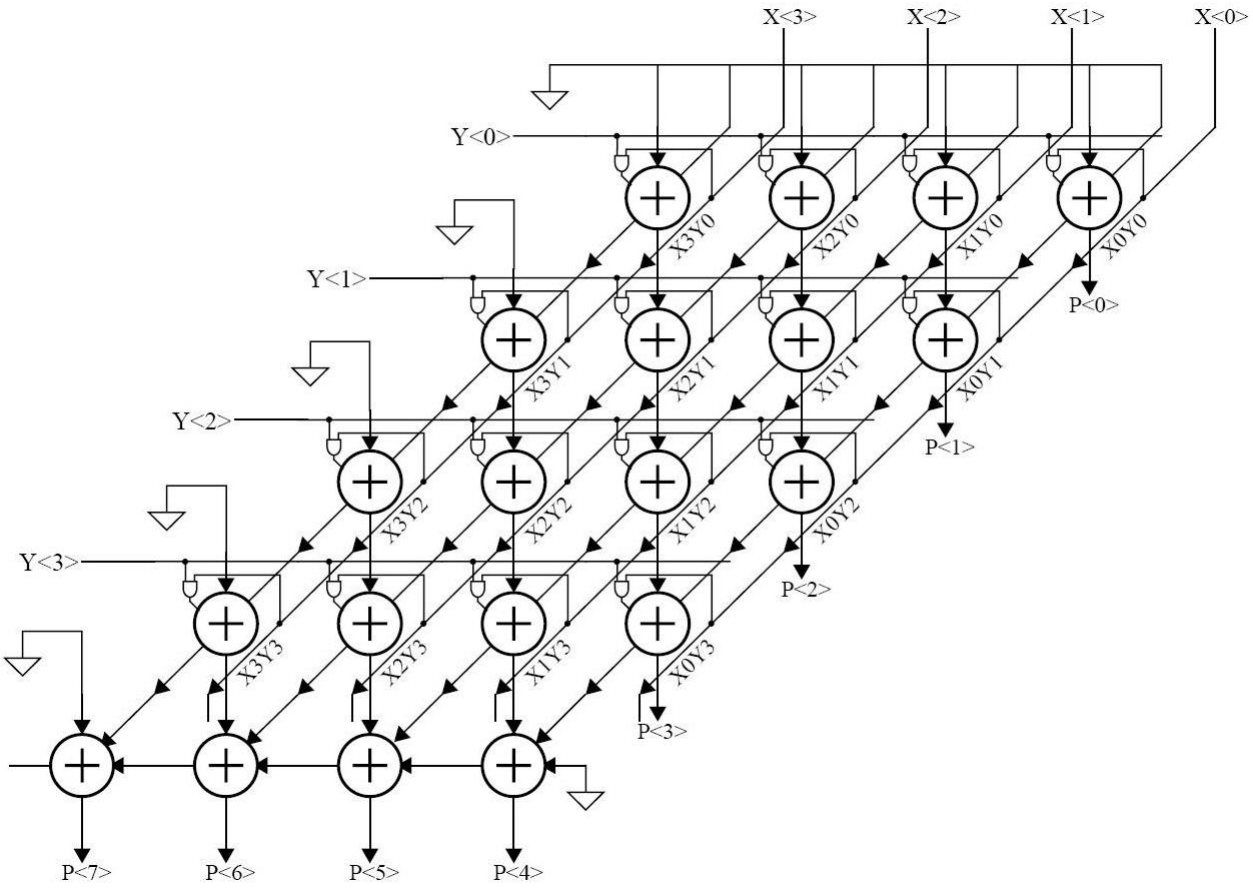


Figure 4.17: Array multiplier

4.4.2 Wallace Tree Multiplication

If the truth table for an adder, is examined, it may be seen that an adder is in effect a \one's counter" that counts the number of l's on the A, B, and C inputs and encodes them on the SUM and CARRY outputs.

A l-bit adder provides a 3:2 (3 inputs, 2 outputs) compression in the number of bits. The addition of partial products in a column of an array multiplier may be thought of as totaling up the number of l's in that column, with any carry being passed to the next column to the left.

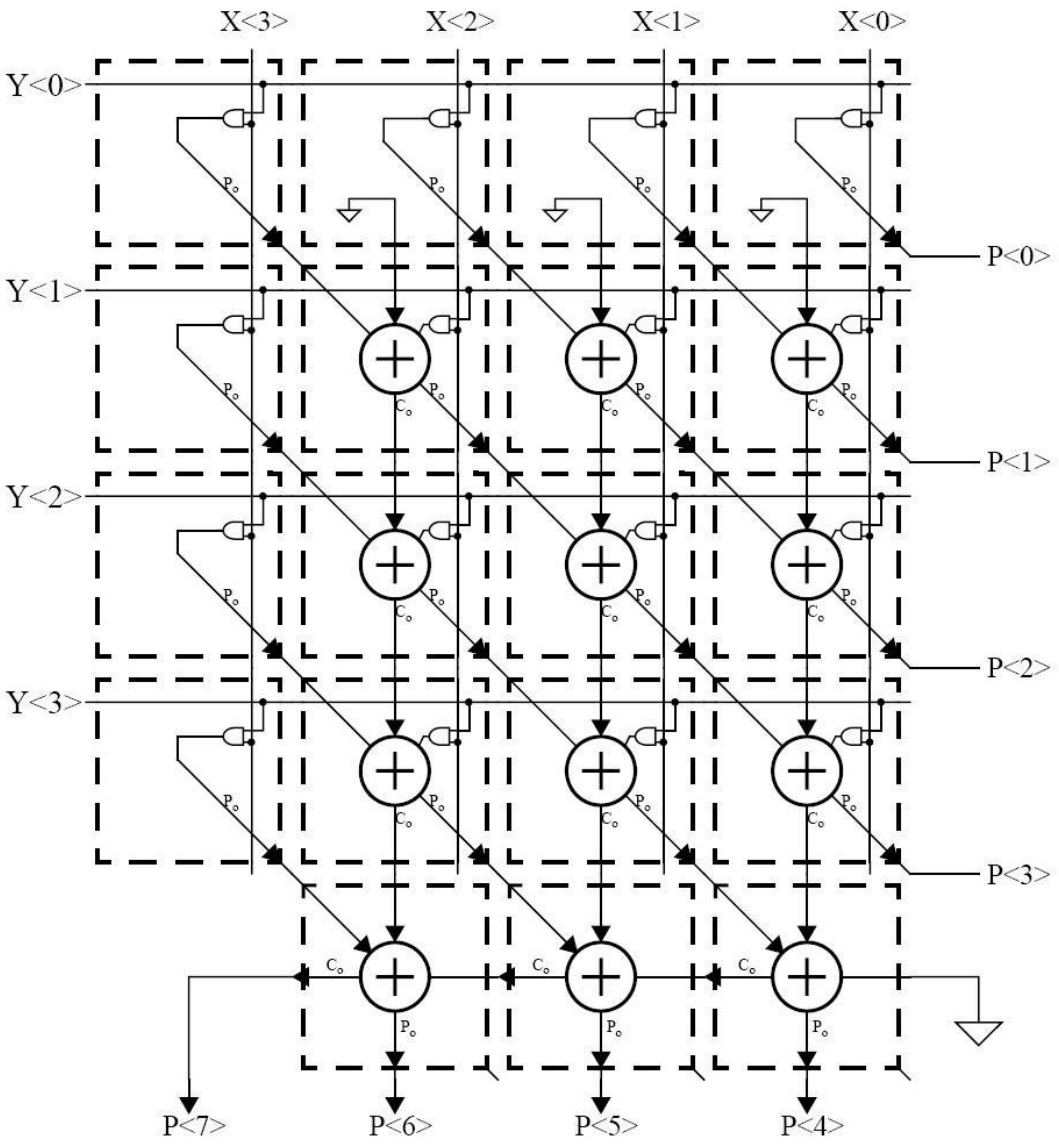


Figure 4.18: Most convenient way for implementation of array multiplier

ABC	Carry/Sum	Number of 1's
0 0 0	0 0	0
0 0 1	1 0	1
0 1 0	1 0	1
0 1 1	0 1	2
1 0 0	0 1	1
1 0 1	1 0	2
1 1 0	1 0	2
1 1 1	1 1	3

Figure 4.19

Example for implementation of 4x4 multiplier (4-bit) using Wallace Tree Multi-plication methods

				X3	X2	X1	X0	Multiplicand
				Y3	Y2	Y1	Y0	Multiplier
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 4.20: Table to nd product terms

Considering the product P3, it may be seen that it requires the summation of four partial products and a possible column carry from the summation of P2.

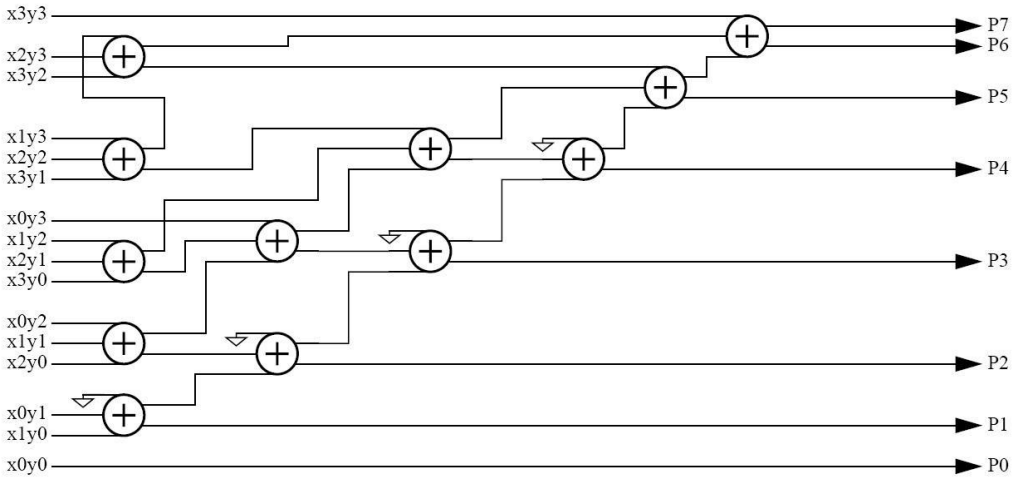


Figure 4.21: Wallace Tree Multiplication for 4-bits

Example for implementation of 6X6 multiplier (4-bit) using Wallace Tree Multi-plication methods

Consider the 6 x 6 multiplication table shown below. Considering the product P5, it may be seen that it requires the summation of six partial products and a possible column carry from the summation of P4. Here we can see the adders required in a multiplier based on this style of addition.

The adders have been arranged vertically into ranks that indicate the time at which the adder output becomes available. While this small example shows the general Wallace addition technique, it does not show the real speed advantage of a Wallace tree. There is an identity table \array part", and a CPA part, which is at the top right. While this has been shown as a ripple-carry adder, any fast CPA can be used here.

						X5	X4	X3	X2	X1	X0	Multiplicand
						Y5	Y4	Y3	Y2	Y1	Y0	Multiplier
						X5Y0	X4Y0	X3Y0	X2Y0	X1Y0	X0Y0	
					X5Y1	X4Y1	X3Y1	X2Y1	X1Y1	X0Y1		
				X5Y2	X4Y2	X3Y2	X2Y2	X1Y2	X0Y2			
			X5Y3	X4Y3	X3Y3	X2Y3	X1Y3	X0Y3				
		X5Y4	X4Y4	X3Y4	X2Y4	X1Y4	X0Y4					
	X5Y5	X4Y5	X3Y5	X2Y5	X1Y5	X0Y5						
P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 4.22: 6 x 6 multiplication table

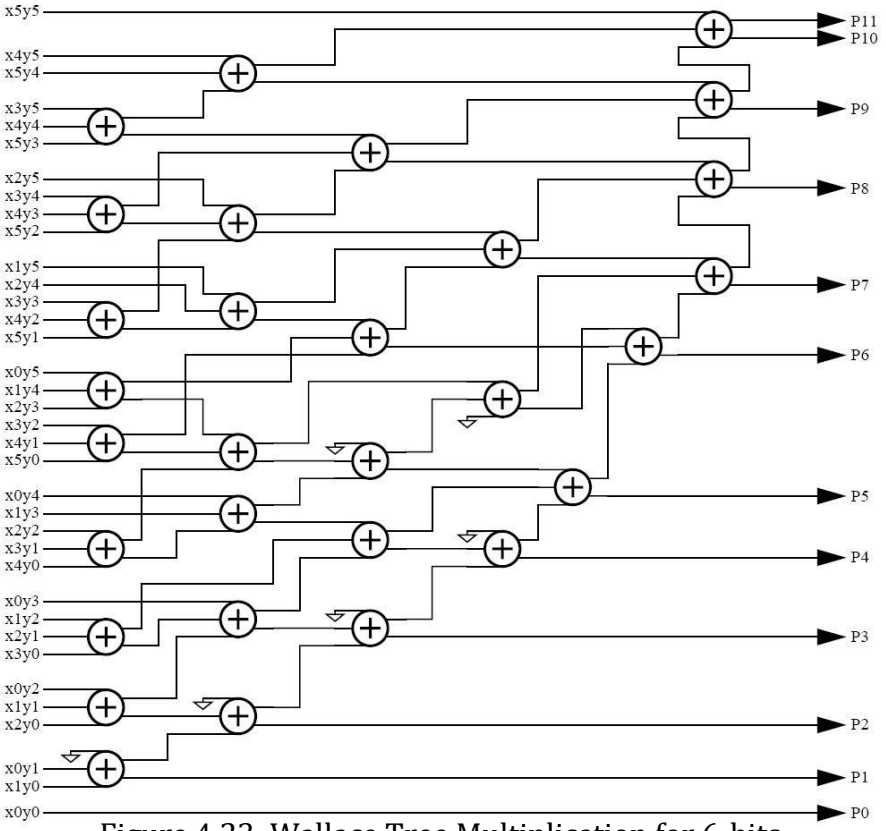


Figure 4.23: Wallace Tree Multiplication for 6-bits

The delay through the array addition (not including the CPA) is proportional to log1.5(n), where n is the width of the Wallace tree.

4.4.3 Baugh-Wooley multiplier:

In signed multiplication the length of the partial products and the number of partial products will be very high. So an algorithm was introduced for signed multiplication called as Baugh-Wooley algorithm. The Baugh-Wooley multiplication is one amongst the cost-effective ways to handle the sign bits. This method has been developed so as to style regular multipliers, suited to 2's compliment numbers.

Let two n-bit numbers, number (A) and number (B), A and B are often pictured as

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \quad (1)$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \quad (2)$$

Where ai and bi area unit the bits during A and B, severally and $an-1$ and $bn-1$ area unit the sign bits. The full precision product, P = A × B, is provided by the equation:

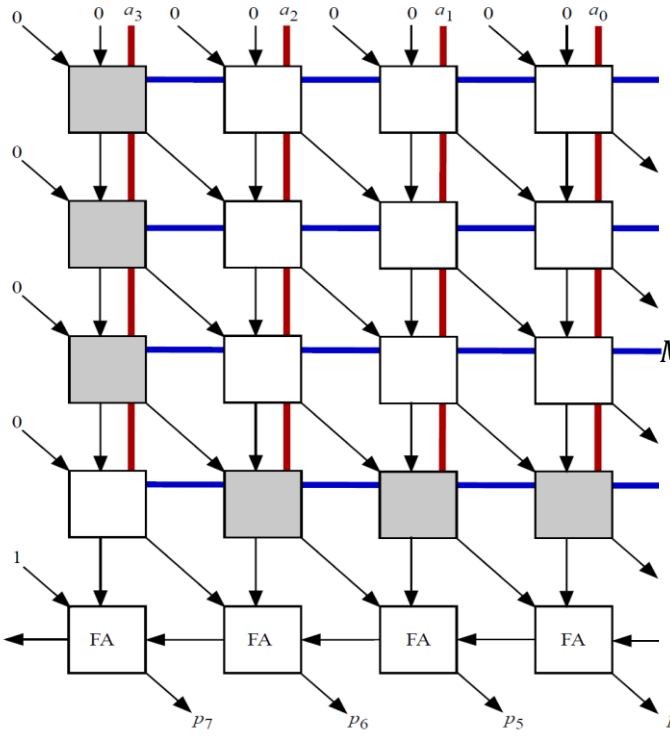
$$P = A \times B = \left[\left(-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i\right) \times \left(-b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i\right)\right]$$

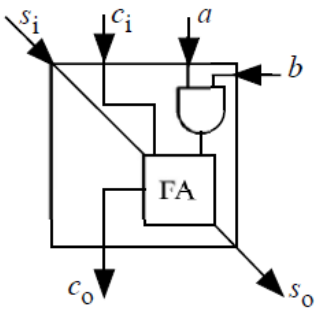
$$= a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-1} a_i 2^i \sum_{j=0}^{n-2} b_j 2^j - 2^{n-1}\sum_{i=0}^{n-2} a_i b_{n-1} 2^i - 2^{n-1}\sum_{j=0}^{n-2} a_{n-1}b_j 2^j \quad (3)$$

The first two terms of above equation are positive and last two terms are negative. In order to calculate the product, instead of subtracting the last two terms, it is possible to add the opposite values. The above equation signifies the Baugh-Wooley algorithm for multiplication process in two's compliment form.

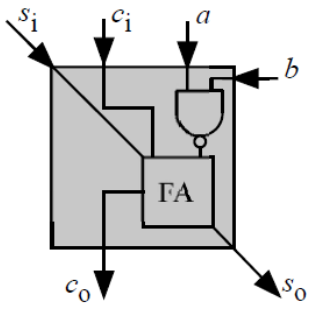
Baugh-Wooley Multiplier provides a high speed, signed multiplication algorithm. It uses parallel products to complement multiplication and adjusts the partial products to maximize the regularity of multiplication array. When number is represented in two's complement form, sign of the number is embedded in Baugh-Wooley multiplier. This algorithm has the advantage that the sign of the partial product bits are always kept positive so that array addition techniques can be directly employed. In the two's complement multiplication, each partial product bit is the AND of a multiplier bit and a multiplicand bit, and the sign of the partial product bits are positive.

4-by-4 Baugh-Wooley multiplier

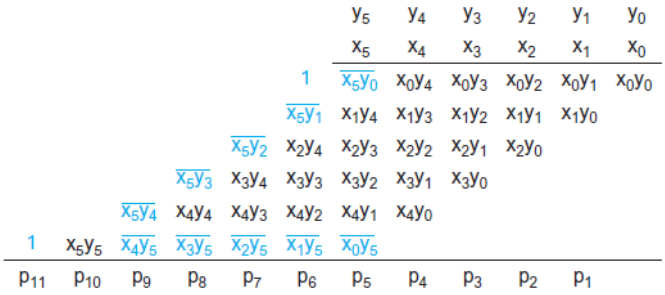


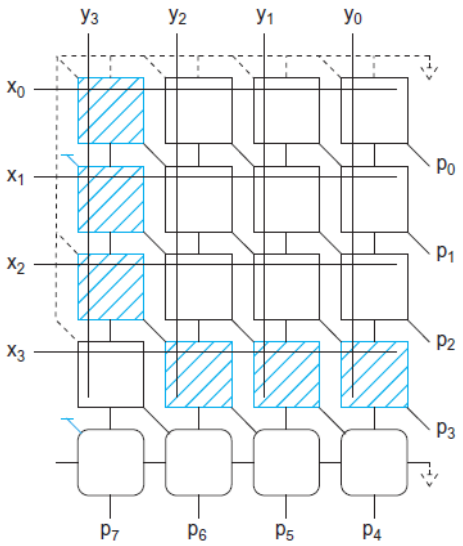


Multiplier white cell



Multiplier Grey cell





-12 $-12_{10} = 10100_2$

X 12 $12_{10} = 01100_2$

-144

By Sign Extension method,

```
    10100
  X 01100
  ---------
  000000000
  00000000
  1110100
  110100
  00000
  ---------
  101110000   ------------>   -144
```

According to the sign extend and invert algorithm,

```
    10100
  X 01100
  ---------
    100000
    10000
   00100
  00100
  10000
  ---------
  101110000   ------------>   - 144
```

4.4.4 Booth Multiplier:

Booth's Algorithm is a smart move for multiplying signed numbers. It initiate with the ability to both add and subtract there are multiple ways to compute a product. Booth's algorithm is a multiplication algorithm that utilizes two's complement notation of signed binary numbers for multiplication.

When multiplying by 9:

- Multiply by 10 (easy, just shift digits left)
- Subtract once

E.g.123454 x 9 = 123454 x (10 – 1) = 1234540 – 123454

- Converts addition of six partial products to one shift and one subtraction

Booth's algorithm applies same principle

- Except no '9' in binary, just '1' and '0'
- So, it's actually easier!

BOOTH ENCODER Booth multiplier reduce the number of iteration step to perform multiplication as compare to conventional steps. Booth Algorithm Scans the multiplier operand and spikes chains of this algorithm can. This algorithm can reduce the number of addition required to produce the result compare to conventional multiplication method. With the help of this algorithm reduce the number of partially product generated in multiplication process by using the modified booth algorithm. Based on the multiplier bits, the process of encoding the multiplicand is performed by radix-4 booth encoder. This recoding algorithm is used to generate efficient partial product.

RADIX-4 BOOTH MULTIPLIER The Radix-4 modified Booth algorithm overcomes all these limitations of Radix-2 algorithm. For operands equal to or greater than 16 bits, the modified Radix-4 Booth algorithm has been widely used. It is based on encoding the two's complement multiplier in order to reduce the number of partial products to be added to n/2.

In Radix-4 Modified Booth algorithm, the number of partial products reduced by half. For multiplication of 2's complement numbers, the two bit encoding using this algorithm scans a triplet of bits. To Booth recode the multiplier term, consider the bits in blocks of three, such that each block overlaps the previous block by one bit. Grouping starts from the LSB, and the first block only uses two bits of the multiplier.

Example

Using Booth algorithm multiply A and B.

A= 20

B=30

A= 0010100 } Please note that both numbers are extended to cover 2A or 2B and the
B= 0011110 } sign bit (whichever is larger).

A * B =

A= 0 0 1 0 1 0 0

B= 0 0 1 1 1 1 0 0

-0

+2 -2

2A = 40 = 00101000

-2A = 11011000

Now performing the addition we have

```
1 1 1 1 1 1 1 0 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 1 0 1 0 0 0
-----------------------
0 0 0 1 0 0 1 0 1 1 0 0 0
```

512 + 64 + 16 + 8 = $(600)_{10}$

Block	Recoded Digit	Operation on X
000	0	0X
001	+1	+1X
010	+1	+1X
011	+2	+2X
100	-2	-2X
101	-1	-1X
110	-1	-1X
111	0	0X

Booth's Algorithm for Binary Multiplication Example

Multiply 14 times -5 using 5-bit numbers (10-bit

result). 14 in binary: 01110

-14 in binary: 10010 (so we can add when we need to subtract the

multiplicand) -5 in binary: 11011

Expected result: -70 in binary: 11101 11010

			Multiplier
Step	Multiplicand	Action	upper 5-bits 0,
			lower 5-bits multiplier,
			1 "Booth bit" initially 0
0	01110	Initialization	00000 11011 0
1	01110	10: Subtract Multiplicand	00000+10010=10010
			10010 11011 0
		Shift Right Arithmetic	11001 01101 1
2	01110	11: No-op	11001 01101 1
		Shift Right Arithmetic	11100 10110 1
3	01110	01: Add Multiplicand	11100+01110=01010 (Carry ignored because adding a positive and negative number cannot overflow.)
			01010 10110 1
		Shift Right Arithmetic	
4	01110	10: Subtract Multiplicand	00101 01011 0
			00101+10010=10111
			10111 01011 0
		Shift Right Arithmetic	
5	01110	11: No-op	11011 10101 1
		Shift Right Arithmetic	11011 10101 1
			11101 11010 1

4.5 Parity generator :

1. Parity is a very useful tool in information processing in digital computers to indicate any presence of error in bit information.

2. External noise and loss of signal strength causes loss of data bit information while transporting data from one device to other device, located inside the computer or externally.

3. To indicate any occurrence of error, an extra bit is included with the message according to the total number of 1s in a set of data, which is called parity.

4. If the extra bit is considered 0 if the total number of 1s is even and 1 for odd quantities of 1s in a set of data, then it is called even parity.

5. On the other hand, if the extra bit is 1 for even quantities of 1s and 0 for an odd number of 1s, then it is called odd parity

A parity generator is a combination logic system to generate the parity bit at the transmitting side.

Four bit message $D_3D_2D_1D_0$	Even parity	Odd parity
0000	0	1
0001	1	0
0010	1	0
0011	0	1
1000	1	0
0101	0	1
0110	0	1
0111	1	0
1000	1	0
1001	0	1
1010	0	1
1011	1	0
1100	0	1
1101	1	0
1110	1	0
1111	0	1

Table 1.1: Truth table for generating even and odd parity bit

If the message bit combination is designated as, $D_3D_2D_1D_0$ and Pe, Po are the even and odd parity respectively, then it is obvious from the table that the Boolean expressions of even parity and odd parity are

$$P_e = D_3 \oplus D_2 \oplus D_1 \oplus D_0$$

$$P_o = \overline{(D_3 \oplus D_2 \oplus D_1 \oplus D_0)}$$

The above illustration is given for a message with four bits of information. However, the logic diagrams can be expanded with more XOR gates for any number of bits.

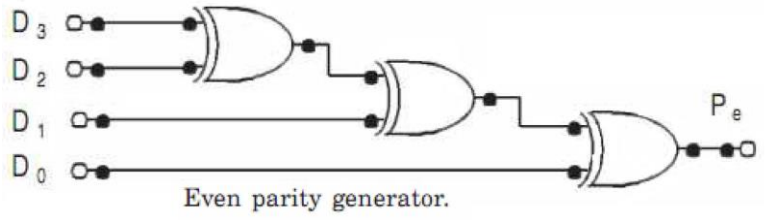


Figure 4.24: Even parity generator using logic gates

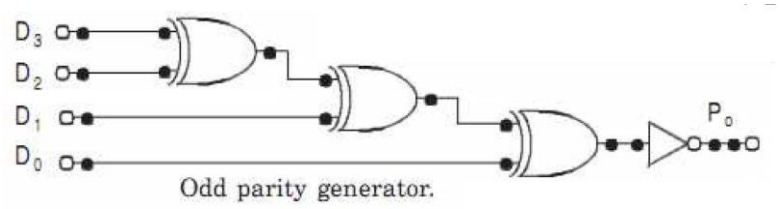


Figure 4.25: Odd parity generator logic gates

4.6 Zero/One detector :

Detecting all ones or zeros on wide N-bit words requires large fan-in AND or NOR gates. Recall that by DeMorgan's law, AND, OR, NAND, and NOR are funda-mentally the same operation except for possible inversions of the inputs and/or outputs. You can build a tree of AND gates, as shown in Figure 4.26(b). Here, alternate NAND and NOR gates have been used. The path has log N stages.

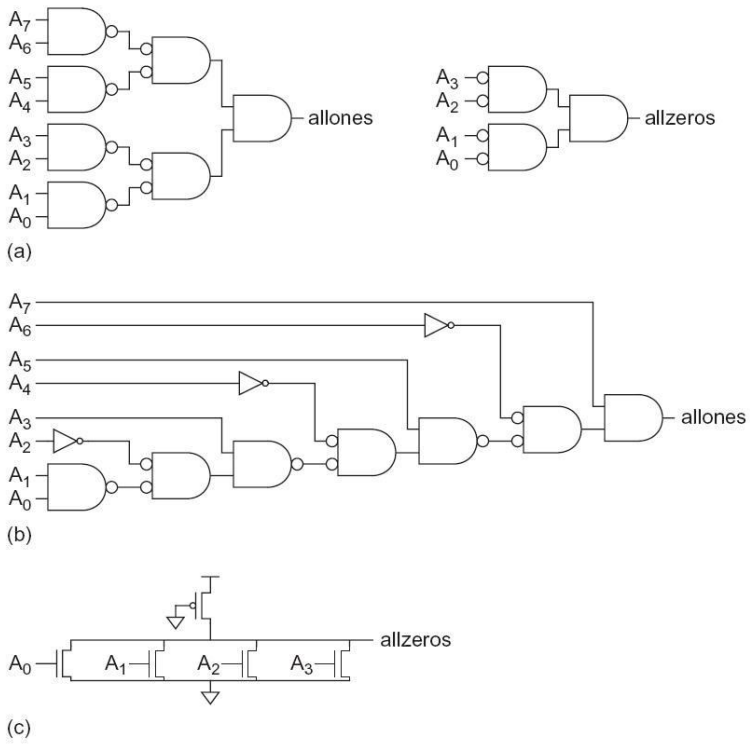


Figure 4.26: One/zero detectors (a) All one detector (b) All zero detector (c) All zero detector transistor level representation

4.7 Comparators :

Another common and very useful combinational logic circuit is that of the Digital Comparator circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean Algebra. There are two main types of Digital Comparator available and these are.

1. Identity Comparator an Identity Comparator is a digital comparator that has only one output terminal for when A = B either HIGH" A = B = 1or LOW" A = B = 0

2. Magnitude Comparator : a Magnitude Comparator is a type of digital com-parator that has three output terminals, one each for equality, A = B greater than,A > B and less than A < B

The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A (A1, A2, A3, . An, etc) against that of a constant or unknown value such as B (B1, B2, B3, . Bn, etc) and produce an output condition or ag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$$A > B; A + B; A < B$$

Which means: A is greater than B, A is equal to B, and A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

Then the operation of a 1-bit digital comparator is given in the following Truth Table.

Inputs		Outputs		
B	A	A > B	A=B	A < B
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	0	0

From the above table the obtained expressions for magnitude comparator using K-map are as follows

For A < B : C = $\overline{A}\,B$

For A = B : D = $\overline{\overline{A}\,B + A\,\overline{B}}$

For A > B : E = $A\overline{B}$ The logic diagram of 1-bit comparator using basic gates is shown below in Figure 4.24.

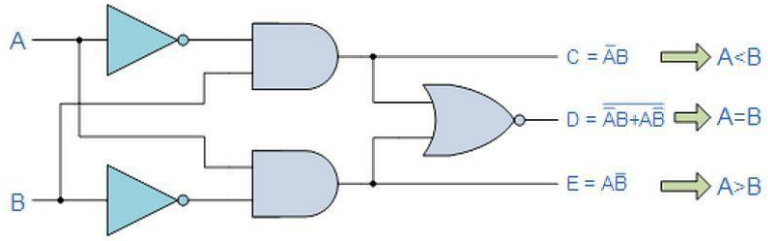
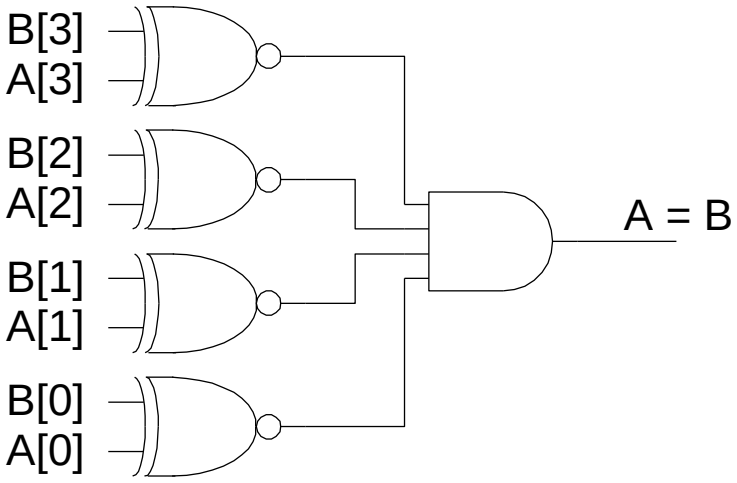


Figure 4.27: 1-bit Digital Comparator

*** Draw separate diagrams for grater, equality and less than expressions.

Equality Comparator:

- Check if each bit is equal (XNOR, aka equality gate)
- 1's detect on bitwise equality



Signed comparison:

For signed numbers, comparison is harder

- C: carry out
- Z: zero (all bits of A-B are 0)
- N: negative (MSB of result)
- V: overflow (inputs had different signs, output sign ≠ B)

Relation	Unsigned Comparison	Signed Comparison
$A = B$	Z	Z
$A \neq B$	$\bar{Z}$	$\bar{Z}$
$A < B$	$C \cdot \bar{Z}$	$\bar{S} \cdot \bar{Z}$
$A > B$	C	S
$A \leq B$	C	$\bar{S}$
$A \geq B$	$\bar{C} + Z$	$S + Z$

Magnitude Comparator:-

* Magnitude comparator, compares two numbers and determines whether a number is greater, smaller or equal to the other given number.

* To compare two numbers A and B, compute $B-A = B+\bar{A}+1$.

Example:

Case-1 $A > B$

$A = 1\ 1\ 1\ 0$
$B = 1\ 1\ 0\ 1$

$\bar{A} = 0\ 0\ 0\ 1$
$B = 1\ 1\ 0\ 1$
$+\quad 1$
Cout $\rightarrow 0 \leftarrow 1\ 1\ 1\ 1$
$Z = 0$

Case-2 $A < B$

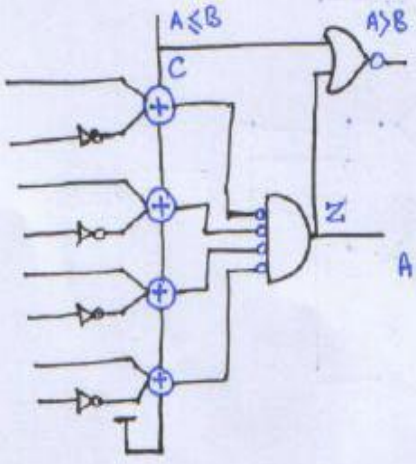
$A = 1\ 1\ 0\ 0$
$B = 1\ 1\ 1\ 1$

$\bar{A} = 0\ 0\ 1\ 1$
$B = 1\ 1\ 1\ 1$
$+\quad 1$
Cout $\leftarrow 1 \leftarrow 0\ 0\ 1\ 1$
$Z = 0$

Case-3 $A = B$

$A = 1\ 1\ 0\ 1$
$B = 1\ 1\ 0\ 1$

$\bar{A} = 0\ 0\ 1\ 0$
$B = 1\ 1\ 0\ 1$
$+\quad 1$
Cout $\leftarrow 1 \leftarrow 0\ 0\ 0\ 0$
$Z = 1$



Relation	Comparison
$A=B$	Z
$A\neq B$	$\bar{Z}$
$A>B$	$\bar{C}$
$A\leq B$	C

4.8 Counters :

Counters can be implemented using the adder/subtractor circuits and registers (or equivalently, D ip- ops)

The simplest counter circuits can be built using T ip- ops because the tog-gle feature is naturally suited for the implementation of the counting operation. Counters are available in two categories

1. Asynchronous(Ripple counters) Asynchronous counters, also known as ripple counters, are not clocked by a common pulse and hence every ip- op in the counter changes at di erent times. The ip- ops in an asynchronous counter is usually clocked by the output pulse of the preceding ip- op. The rst ip- op is clocked by an external event.

   The ip- op output transition serves as a source for triggering other ip- ops i.e the C input (clock input) of some or all ip- ops are triggered NOT by the common clock pulses
   Eg:- Binary ripple counters, BCD ripple counters
2. Synchronous counters A synchronous counter however, has an internal clock, and the external event is used to produce a pulse which is synchronized with this internal clock.

   C input (clock input) of all ip- ops receive the common clock pulses

   E.g.:- Binary counter, Up-down Binary counter, BCD Binary counter, Ring counter, Johnson counter,

4.8.1 Asynchronous Up-Counter with T Flip-Flops

Figure 4.28 shows a 3-bit counter capable of counting from 0 to 7. The clock inputs of the three ip- ops are connected in cascade. The T input of each ip-op is connected to a constant 1, which means that the state of the ip- op will be toggled at each active edge (here, it is positive edge) of its clock. We assume that the purpose of this circuit is to count the number of pulses that occur on the primary input called Clock. Thus the clock input of the rst ip- op is connected to the Clock line. The other two ip- ops have their clock inputs driven by the $\overline{Q}$ output of the preceding ip- op. Therefore, they toggle their states whenever the preceding ip- op changes its state from Q = 1 to Q = 0, which results in a positive edge of the $\overline{Q}$ signal.

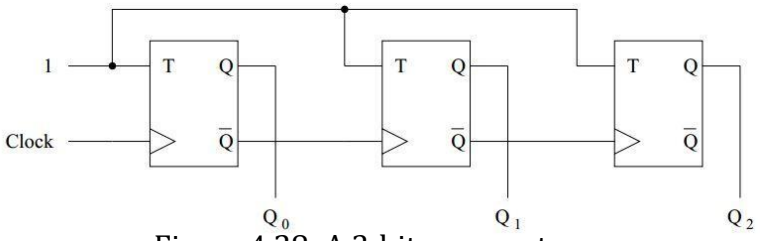


Figure 4.28: A 3-bit up-counter.

Note here the value of the count is the indicated by the 3-bit binary number Q2Q1Q0. Since the second ip- op is clocked by $\overline{Q_0}$, the value of Q_1 changes shortly after the change of the $\overline{Q_0}$ signal. Similarly, the value of $\overline{Q_2}$ changes shortly after the change of the $\overline{Q_1}$ signal. This circuit is a modulo-8 counter. Because it counts in the upward direction, we call it an up-counter. This behavior is similar to the rippling of carries in a ripple-carry adder. The circuit is therefore called an asynchronous counter, or a ripple counter.

4.8.2 Asynchronous Down-Counter with T Flip-Flops

Some modifications of the circuit in Figure 4.29 lead to a down-counter which counts in the sequence 0, 7, 6, 5, 4, 3, 2, 1, 0, 7, and so on. The modified circuit is shown in Figure 3. Here the clock inputs of the second and third ip- ops are driven by the Q outputs of the preceding stages, rather than by the Q outputs.

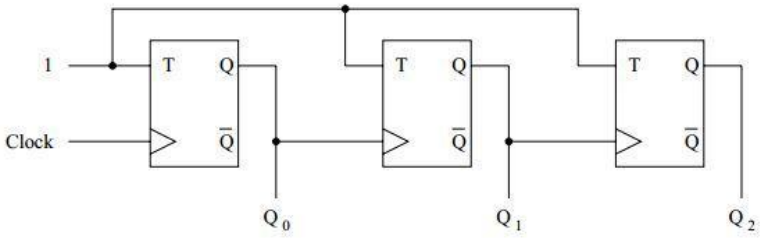


Figure 4.29: A 3-bit down-counter.

Although the asynchronous counter is easier to construct, it has some major disadvantages over the synchronous counter.

First of all, the asynchronous counter is slow. In a synchronous counter, all the ip- ops will change states simultaneously while for an asynchronous counter, the propagation delays of the ip- ops add together to produce the overall delay. Hence, the more bits or number of ip-ops in an asynchronous counter, the slower it will be.

4.8.3 Synchronous Counters

A synchronous counter usually consists of two parts: the memory element and the combinational element. The memory element is implemented using ip- ops while the combinational element can be implemented in a number of ways. Using logic gates is the traditional method of implementing combinational logic and has been applied for decades.

4.8.4 Synchronous Up-Counter with T Flip-Flops

An example of a 4-bit synchronous up-counter is shown in Figure 5. Observing the

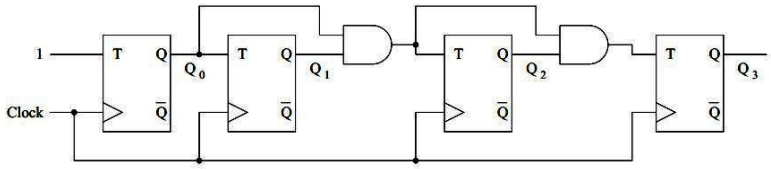


Figure 4.30: A 4bit synchronous upcounter

Clock Cycle	Q_3	Q_2	Q_1	Q_0	
0	0	0	0	0	Q_1 changes
1	0	0	0	1	Q_2 changes
2	0	0	1	0	
3	0	0	1	1	Q_3 changes
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	
16	0	0	0	0	

Figure 4.31: Contents of a 4bit upcounter for 16 consecutive clock cycles

pattern of bits in each row of the table, it is apparent that bit Q0 changes on each clock cycle. Bit QQ_1 changes only when $Q_0 = 1$. Bit Q_2 changes only when both Q_1 and Q_0 are equal to 1. Bit Q_3 changes only when $Q_2 = Q_1 = Q_0 = 1$. In general, for an n-bit up-counter, a give ip- op changes its state only when all the preceding ip- ops are in the state Q = 1. Therefore, if we use T ip- ops to realize the 4-bit counter, then the T inputs should be de ned as

$$T_0 = 1$$

$$T_1 = Q_0$$

$$T_2 = Q_0Q_1$$

$$T_3 = Q_0Q_1Q_2$$

In Figure 5, instead of using AND gates of increased size for each stage, we use a factored arrangement. This arrangement does not slow down the response of the

counter, because all ip- ops change their states after a propagation delay from the positive edge of the clock. Note that a change in the value of Q0 may have to propagate through several AND gates to reach the ip- ops in the higher stages of the counter, which requires a certain amount of time. This time must not exceed the clock period. Actually, it must be 3less than the clock period minus the setup time of the ip- ops. It shows that the circuit behaves as a modulo-16 up-counter. Because all changes take place with the same delay after the active edge of the Clock signal, the circuit is called a synchronous counter.

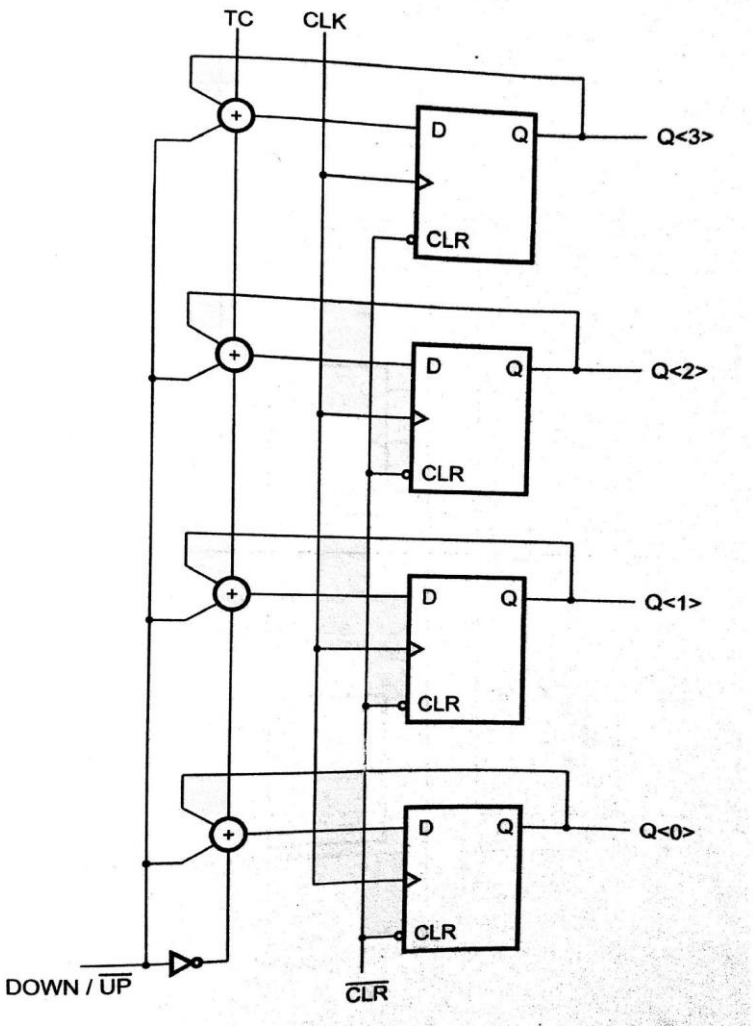


Figure 4.32: Design of synchronous counter using adders and registers

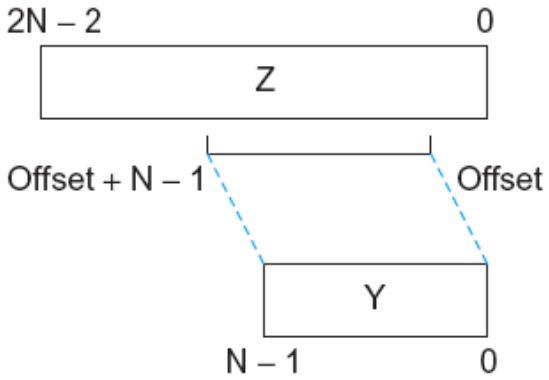
4.9 Shifters

4.9.1 Shifters :

- Logical Shift:
  - Shifts number left or right and fills with 0's
    - 1011 LSR 1 = 0101 1011 LSL1 = 0110
- Arithmetic Shift:
  - Shifts number left or right. Rt shift sign extends
    - 1011 ASR1 = 1101 1011 ASL1 = 0110
- Rotate:
  - Shifts number left or right and fills with lost bits
    - 1011 ROR1 = 1101 1011 ROL1 = 0111

4.9.2 Funnel Shifter

- A funnel shifter can do all six types of shifts
- Selects N-bit field Y from 2N–1-bit input
  - Shift by k bits ($0 \le k < N$)
  - Logically involves N N:1 multiplexers



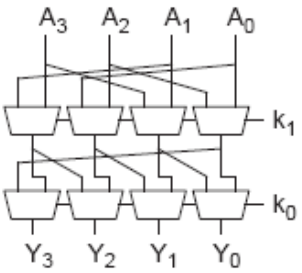
Funnel Source Generator

Shift Type	$Z_{2N-2:N}$	Z_{N-1}	$Z_{N-2:0}$	Offset
Logical Right	$A_{N-2:0}$	A_{N-1}	$A_{N-2:0}$	k
Arithmetic Right	0	A_{N-1}	$A_{N-2:0}$	k
Rotate Right	sign	A_{N-1}	$A_{N-2:0}$	k
Logical/Arithmetic Left	$A_{N-1:1}$	A_0	$A_{N-1:1}$	$\overline{k}$
Rotate Left	$A_{N-1:1}$	A_0	0	$\overline{k}$

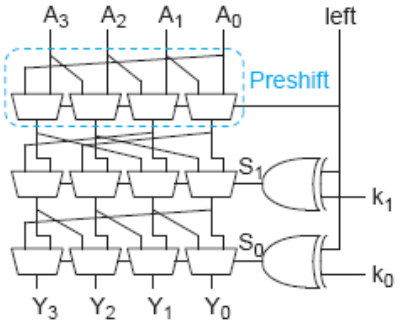
4.9.3 Barrel Shifter

- Barrel shifters perform right rotations using wrap-around wires.
- Left rotations are right rotations by N – k = k + 1 bits.
- Shifts are rotations with the end bits masked off.

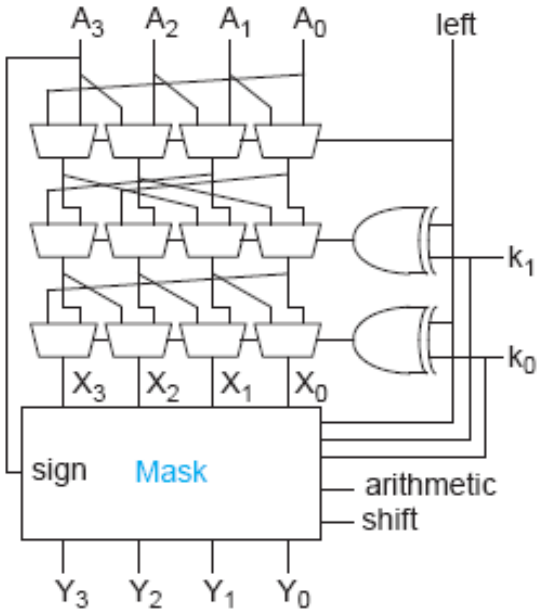
Logarithmic Barrel Shifter



Right shift only

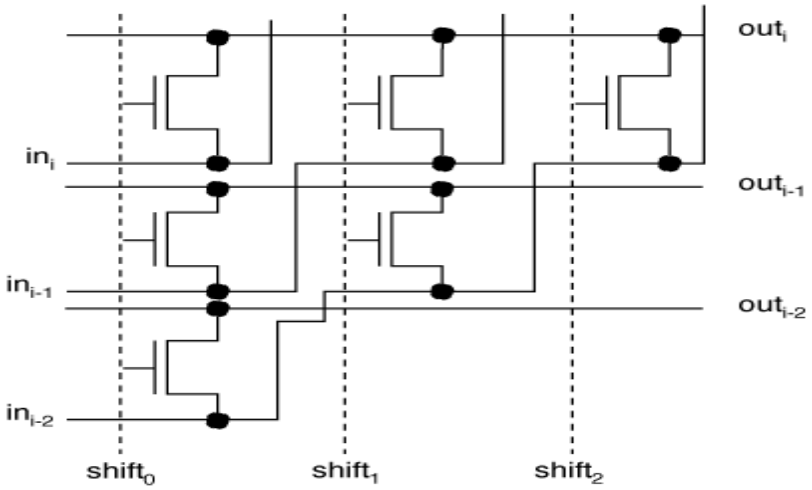


Right/Left shift



Right/Left Shift & Rotate

Barrel shifter cell



4.9.4

LFSR (Linear Feedback Shift Register):-

* LFSR is an example of PRSGs (Pseudo Random Sequence Genelators) that are used to genelate random number sequences and their special applications are stimuli generation in BIST (Built-In-Self-Test) and other testability techniques.

* LFSR consists of N registers connected together as a shift register.

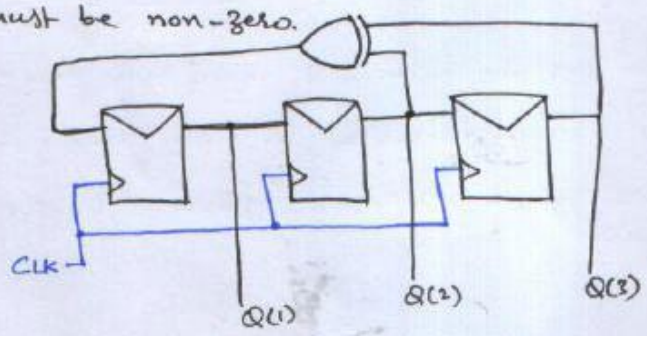
* The input to shift register comes from the XOR of particular bits of the register. The bits that are fed to XOR are called tap sequence and are often specified with characteristic polynomial. Some characteristic polynomial are shown on next page.

* For LFSR to work as PRSG, it must be initialized to a non-zero seed value.

* An LFSR is considered maximal LFSR if it genelates $2^n - 1$ distinct sequences.

* For maximal LFSR, the following conditions must be met:

  1. Bits in the tap sequence must be even.
  2. n-th bit of register must be part of XOR operation i-e tap sequence.
  3. Feedback must be given to the first bit
  4. Seed value must be non-zero.



3-bit figure (LFSR)

* Truth table for 3-bit LFSR is given below:

cycle	Q[1]	Q[2]	Q[3]
0	1	1	0
1	1	1	1
2	0	1	1
3	0	0	1
4	1	0	0
5	0	1	0
6	1	0	1
7	1	1	0

→ 110 is seed value

* Observe: after 0th cycle i.e seed value,

$Q[1] = Q[2] \oplus Q[3]$

$Q[2] = Q[1]$ → Shift

$Q[3] = Q[2]$

FEEDBACK

→ sequence 110 repeated

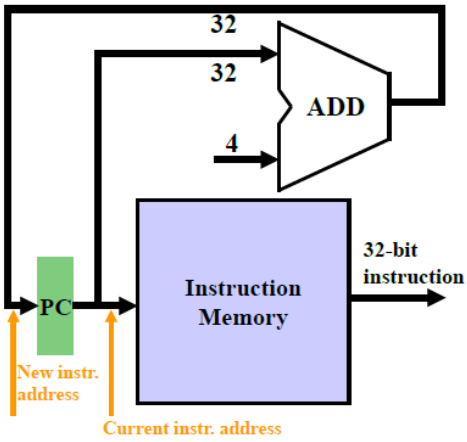
* Example characteristic polynomials for Maximal LFSRs of various sizes are given below:

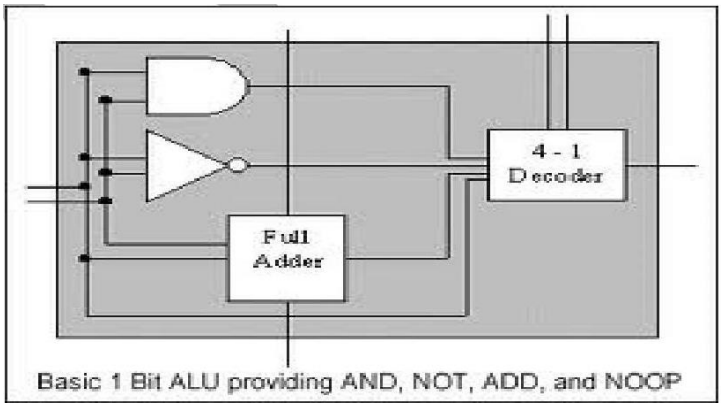
N	Polynomial
3	$1+x^2+x^3$
4	$1+x^3+x^4$
5	$1+x^3+x^5$
6	$1+x^5+x^6$
7	$1+x^6+x^7$
8	$1+x^1+x^6+x^7+x^8$
9	$1+x^5+x^9$
15	$1+x^{14}+x^{15}$
16	$1+x^4+x^{13}+x^{15}+x^{16}$

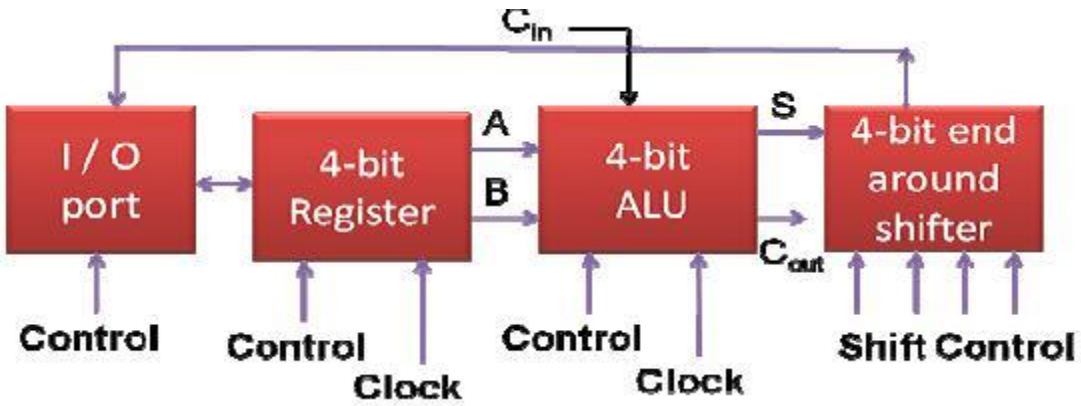
* For certain lengths, N, more than two taps may be required.
* For many values of N, there are multiple polynomials resulting in different maximal lengths LFSRs.

4.10 ALU:

An ALU is a Arithmetic Logic Unit that requires Arithmetic operations and Boolean operations. Basically arithmetic operations are addition and subtraction. one may either multiplex between an adder and a Boolean unit or merge the Boolean unit into the adder as in tha classic transistor-transistor logic.





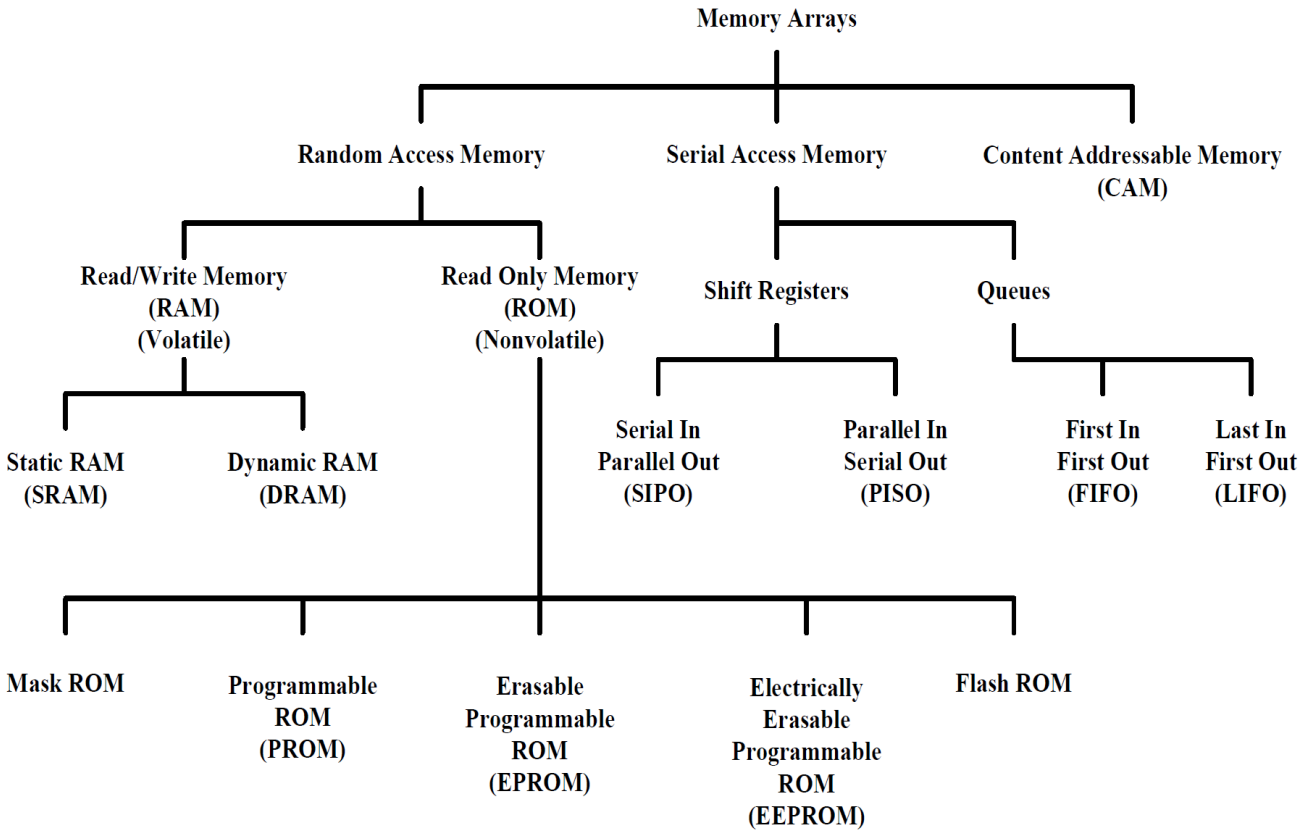


4-bit data path for processor

The heart of the ALU is a 4-bit adder circuit. A 4-bit adder must take sum of two 4-bit numbers, and there is an assumption that all 4-bit quantities are presented in parallel form and that the shifter circuit is designed to accept and shift a 4-bit parallel sum from the ALU. The sum is to be stored in parallel at the output of the adder from where it is fed through the shifter and back to the register array. Therefore, a single 4-bit data bus is needed from the adder to the shifter and another 4-bit bus is required from the shifted output back to the register

4.11 Memory Array

The memory array is classified into 3 types - Random Access memory (RAM), Serial access memory and content addressable memory (CAM). We will discuss each type in detail.



4.12 Read only memory (ROM)

The basic idea of the memory that can only be read and never altered is called Read only memories. There are vast and variety of potential applications for these kind of memories. Programs for processors with fixed applications such as washing machines, calculators and game machines, once developed and debugged, need only reading. Fixing the contents at manufacturing time leads to small and fast implementation.

There are different ways to implement the logic of ROM cells, the fact that the contents of a ROM cell are permanently fixed considerably simplifies its design. The cell should be designed so that a „0" or „1" is presented to the bitline upon activation of its"wordline. The different approaches for implementing the ROM cells are Diode ROM, MOS ROM 1 and MOS ROM 2. These are the main approaches for designing a larger density ROMs.

4.12.1 Mask ROM :

The ROM memories which we have seen earlier are application specific ROMs where the memory module is part of a larger custom design and programmed for that particular application only. The ROMs which we are going to discuss in this section are commodity ROMs, where a vendor mass-produces memory modules that are later customized according to customer specifications. Under these circumstances, it is essential that the number of process steps involved in programming be minimal and that they can be performed as a last phase of the manufacturing process. In this way large amounts of programmed dies can be preprocessed.

This mask-programmable approach preferably uses the contact mask to personalize or program the memory. The programming of a ROM module involves the manufacturer, which introduces an unwelcome delay in product development. The major usage of this ROM was in system-on-a-chip where the majority of the chip is preprocessed, only the minor part of the die is mask programmed. The other usages of this ROM are to program the microcontroller, embedded on the chip, for a variety of applications.

NOR-based ROM

The building block of this ROM is a pseudo-nMOS NOR gate as in Figure 4.33

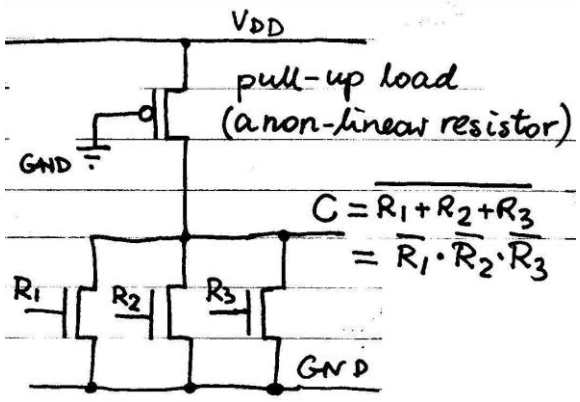
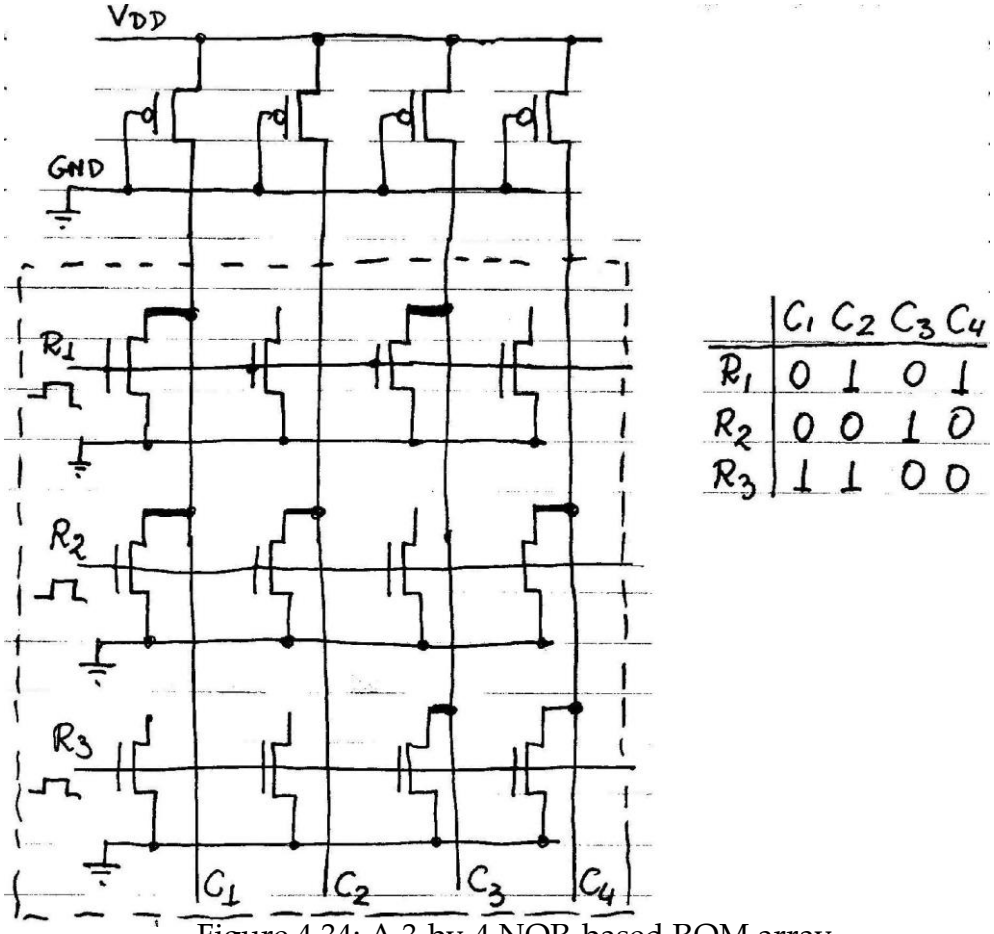


Figure 4.33: A 3-input pseudo-nMOS NOR gate.

Unlike in a standard CMOS gate, the pMOS pull-up circuitry is replaced by a single pMOSwith its gate tied up to GND, hence being permanently on acting as a load resistor.
If none of the nMOS transistors is activated (all R_i being low) then the output signal C is high.
If any of the nMOS transistors is activated (R_i being high) then the output signal C is low.
To reduce the power consumption the gate of the pMOS pull-up transistor is connected to a clock signal. The power is consumed only during low period of the clock.

NOR-based ROM consists of m n-input pseudo-nMOS NOR gates, one n-input NOR per column as shown in Figure 4.34.



	C_1	C_2	C_3	C_4
R_1	0	1	0	1
R_2	0	0	1	0
R_3	1	1	0	0

Figure 4.34: A 3-by-4 NOR-based ROM array

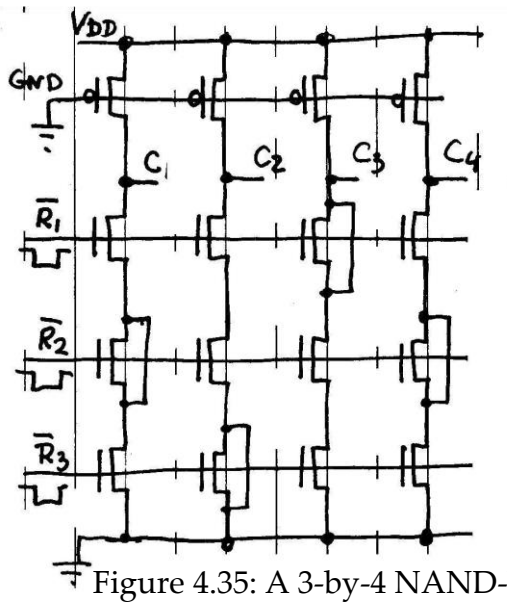
Each memory cell is represented by one nMOS transistor and a binary information is stored by connecting or not the drain terminal of such a transistor to the bit line.

For every row address only one word line is activated by applying a high signal to the gates of nMOS transistors in a row.

If a selected transistor in the i-th column is connected to a bit line then the logic '0' is stored in this memory cell. if the transistor is not connected, then the logic '1' is stored.

NAND-based ROM

A NAND-based ROM consists of m n-input pseudo-nMOS NAND gates, one n-input NAND per column as shown in Figure 4.35. In this case, we have up to n serially connected nMOS transistors in each column.



	C_1	C_2	C_3	C_4
$\overline{R_1}$	1	1	0	1
$\overline{R_2}$	0	1	1	0
$\overline{R_3}$	1	0	1	1

Figure 4.35: A 3-by-4 NAND-based ROM array

For every row address only one word line is activated by applying a low signal to the gates of nMOS transistors in a row. When no word line is activated, all nMOS transistors are on and the line signals, C_i are all low.

When a word line is activated all transistors in the row are switched off and the respective C_i signals are high. If a transistor in the selected row is short-circuited, then the respective C_i signal is low.

In other words, the logic '0' is stored when a transistor is replaced with a wire, whereas the logic '1' is stored by an nMOS transistor being present.

4.12.2 Programmable ROM (PROM) :

The technology that offers its users to program the memory one time is called Programmable ROM. It is also called as WRITE ONCE device. This is most often accomplished by introducing fuses (implemented in nichrome, polysilicon, or other conductors) in the memory cell. During the programming phase, some of these fuses are blown by applying a high current, which disables the connected transistor.

While PROMs have the advantage of being "customer programmable," the single write phase makes them unattractive. For instance, a single error in the programming process or application makes the device unstable. This explains the current preference for devices that can be programmed several times.

The Floating-Gate transistor is the device at the heart of the majority of reprogrammable memories. Various attempts have made to create a device with electrically alterable characteristics and enough reliability to support a multitude of write cycles. The floating gate structure is similar to a traditional MOS device, except that an extra polysilicon strip is inserted between the gate and channel.

This strip is not connected to anything and is called a floating gate. The most obvious impact of inserting this extra gate is to double the gate oxide thickness t_{ox}, which results in a reduced device transconductance as well as an increased threshold voltage. Though these properties are not desirable but from other point of view this device acts as a normal transistor.

The most important property of this device is that the threshold voltage of this device is programmable. By applying a high voltage (above 10V) between the source and the gate-drain terminals creates a high electric field and causes avalanche injection to occur. Electrons acquire sufficient energy to become "hot" and traverse through the first oxide insulator, so that they get trapped on the floating gate. In reference to the programming mechanism, the floating-gate transistor is often called a floating-gate avalanche-injection MOS.

The trapping of electrons on the floating gate effectively drops the voltage on the gate. This process is self-limiting – the negative charge accumulated on the floating gate reduces the electrical field over the oxide so that ultimately it becomes incapable of accelerating any more hot electrons. Virtually all nonvolatile memories are currently based on the floating-gate mechanism. Different classes can be identified, based on the erasure mechanism.

4.12.3 Erasable-programmable Read-Only Memory (EPROM) :

The erasure mechanism in EPROM is based on the shining ultraviolet light on the cells through a transparent window in the package. The UV radiation renders the oxide to conduct by the direct generation of electron-hole pairs in the material. The erasure process is slow depending on the UV source, it can take from seconds to several minutes. The programming takes several μs/word. Alternatively there is another problem which exists is the limited endurance - the number of erase/program cycles is limited to a maximum of one thousand mainly as a result of UV erasing procedure. The device thresholds might vary with repeated programming cycles. The on-chip circuitry is designed in such a way that it also controls the value of the thresholds to within a specified range during programming. The injection of large channel current of 0.5 mA at a control gate voltage of 12.5V causes high power dissipation during programming.

On the other hand, EPROM is extremely simple and dense, making it possible to fabricate large memories at a low cost. Therefore EPROMs were attractive in applications that do not require reprogramming. The major disadvantage of the EPROM is that the erasure procedure has to occur "off system". This means the memory must be removed from the board and placed in an EPROM programmer for programming.

4.12.4 Electrically Erasable Programmable Read-Only Memory EEPROM)

The disadvantage of the EPROM [16] is solved by using a method to inject or remove charges from a floating-gate namely – tunneling. A modified floating-gate device called FLOTOX (floating-gate tunneling oxide) transistor is used as programmable device that supports an electrical-erasure procedure. It resembles FAMOS (floating-gate avalanche MOS) device, except that a portion of the dielectric separating the floating gate from the channel and drain is reduced in thickness to about 10 nm or less.

The main advantage of this programming approach is that it is reversible; that is, erasing is simply achieved by reversing the voltage applied during the writing process. The electrons injection on floating-gate raises the threshold, while the reverse operation lowers the V_T. When a voltage of approximately 10V (equivalent to 10^9 V/m) is applied over the thin insulator, electrons travel to and from the floating gate through a mechanism called Fowler – Nordheim tunneling.

4.12.5 Flash Electrically Erasable Programmable ROM (Flash) :

The concept of Flash EEPROMs is a combination of density of EPROM with versatility of EEPROM structures, with cost and functionality ranging from somewhere between two. Most Flash EEPROM devices use the avalanche hot-electron-injection approach to program the device. Erasure is performed using Fowler – Nordheim tunneling, as from EEPROM cells. The main difference is that erasure procedure is performed in bulk for a complete chip or for the subsection of the memory. Erasing complete memory core at once makes it possible to carefully monitor of the device characteristics during erasure.

The monitoring control hardware on the memory chip regularly checks the value of the threshold during erasure, and adjusts the erasure time dynamically. This approach is only practical when erasing large chunks of memory at a time; hence the flash concept. One of the many existing alternatives for Flash EEPROMs memories are ETOX devices. It resembles a FAMOS gate except that a very thin tunneling gate oxide is utilized (10 nm). Different areas of the gate oxide are used for programming and erasure. Programming is performed by applying a high voltage (12V) on the gate and drain terminals for a grounded source, while erasure occurs with the gate rounded and the source at 12V.

The Programming cycle starts with an erase operation. In erase operation, A 0V gate voltage is applied and a 12V supply is given at source. Electrons, if any, are ejected to the source by tunneling. All cells are erased simultaneously. The variations caused in the threshold voltage at the end of erase operation are due to different initial values of cell threshold voltage and variations in oxide thickness. This can be solved in two methods:

1. The array cells are programmed before applying the erase pulse so that the entire threshold starts at approximately same time.
2. An erase pulse of controlled width is applied. Subsequently the whole array is read to ensure that all the cells are erased. If not another erase pulse is applied followed by the read cycle.

For write (programming) operation, a high voltage is applied to the gate of the selected device. If a „1" is applied to the drain at that time, hot electrons are generated and injected onto the floating gate, raising the threshold. Read operation corresponds as the wordline is raised to 5V; it causes a conditional discharge of bitline.

4.13 **Random Access memory (RAM) :**

Random access memory is a type of computer data storage. It is made of integrated circuits that allow the stored data to be accessed in any order i.e., at random and without the physical movement of storage medium or a physical reading head. RAM is a volatile memory as the information or the instructions stored in the memory will be lost if the power is switched off.

The word "random" refers to the fact that any piece of data can be returned at a constant time regardless of its physical location and whether or not it is related to the previous piece of data. This contrasts with the physical movement devices such as tapes, magnetic disks and optical disks, which rely on physical movement of the recording medium or reading head. In these devices, the retrieval time varies with the physical location and the movement time takes longer than the data transfer.

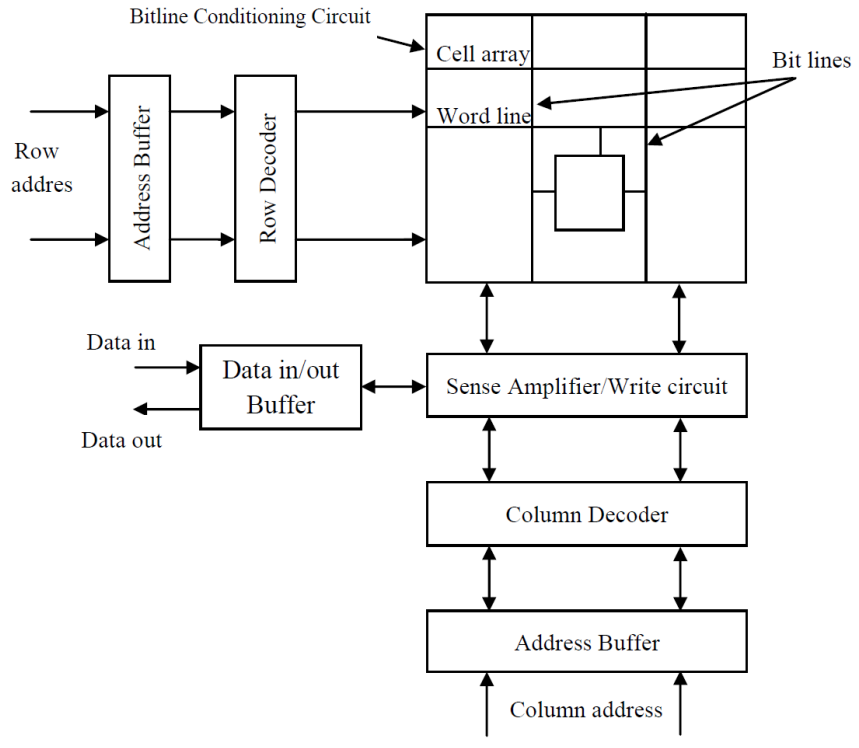
The main advantages of RAM over types of storage which require physical movement is that retrieval times are short and consistent.Short because no physical movement is necessary and consistent the time taken to retrieve the data does not depend on the current distance from a physical head. The access time for retrieving any piece of data in RAM chip is same. The disadvantages are its cost compared to the physical moving media and loss of data when power is turned off.

RAM is used as 'main memory' or primary storage because of its speed and consistency. The working area used for loading, displaying and manipulating applications and data. In most personal computers, the RAM is not an integral part of the motherboard or CPU. It comes in the easily upgraded form of modules called memory sticks. These can quickly be removed and replaced when they are damaged or when the system needs up gradation of memory depending on current purposes. A smaller amount of random-access memory is also integrated with the CPU, but this is usually referred to as "cache" memory, rather than RAM. Modern RAM generally stores a bit of data as either a charge in a capacitor, as in dynamic RAM, or the state of a flip-flop, as in static RAM.

4.14 Static Random Access Memory (SRAM)

4.15 SRAM Architecture:

The typical SRAM design is shown in figure 1.8 the memory array contains the memory cells which are readable and writable. The Row decoder selects from 1 out of n = 2k rows, while the column decoder selects l = 2i out of m = 2j columns. The addresses are not multiplexed as it in the DRAM. Sense amplifier detects small voltage variations on the memory complimentary bitline which reduces the reading time. The conditioning circuit is used to pre-charge the bitlines.



Typical SRAM Architecture

In a read operation, the bitlines are precharged to some reference voltage usually close to the supply voltage. When word line turns high, the access transistor connected to the node storing „0" starts discharging the bitline while the complementary bitline remains in its precharged state, resulting in a differential voltage between the bitline pair.
Since the SRAM has an optimized area results in a small cell current and slow bitline discharge rate. In order to speed up the RAM access, sense amplifiers are used which amplify the small bitline signal and eventually drive it to the external world.
The word "static" means that the memory retains its contents as long as the power is turned on. Random access means that locations in the memory can be written to or read from in any order, regardless of the memory location that was last accessed. Each bit in an SRAM is stored on four transistors that form two cross-coupled inverters. This storage cell has two stable states which are used to denote „0" and „1". The access transistors are used to access the stored bits in the SRAM during read or write mode.

It thus typically takes six MOSFETs to store one memory bit. Access to the cell is enabled by the word line WL which controls the two access transistors N_1 and N_2 which, in turn, control whether the cell should be connected to the bitlines BL and /BL. They are used to transfer data for both read and write operations. The bitlines are complementary as it improves the noise margin. Chapter 2 explains more about SRAMs and its Read/Write operations.

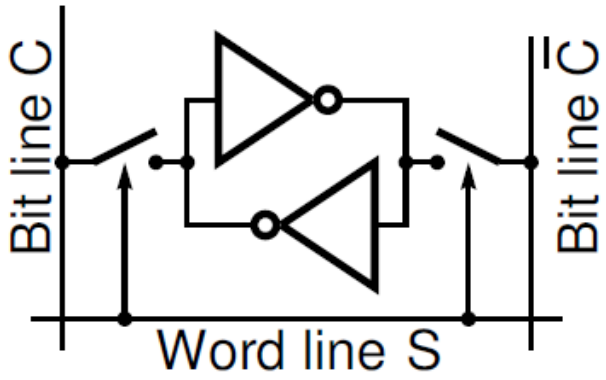


Figure 4.36: A logic diagram of a CMOS static memory cell

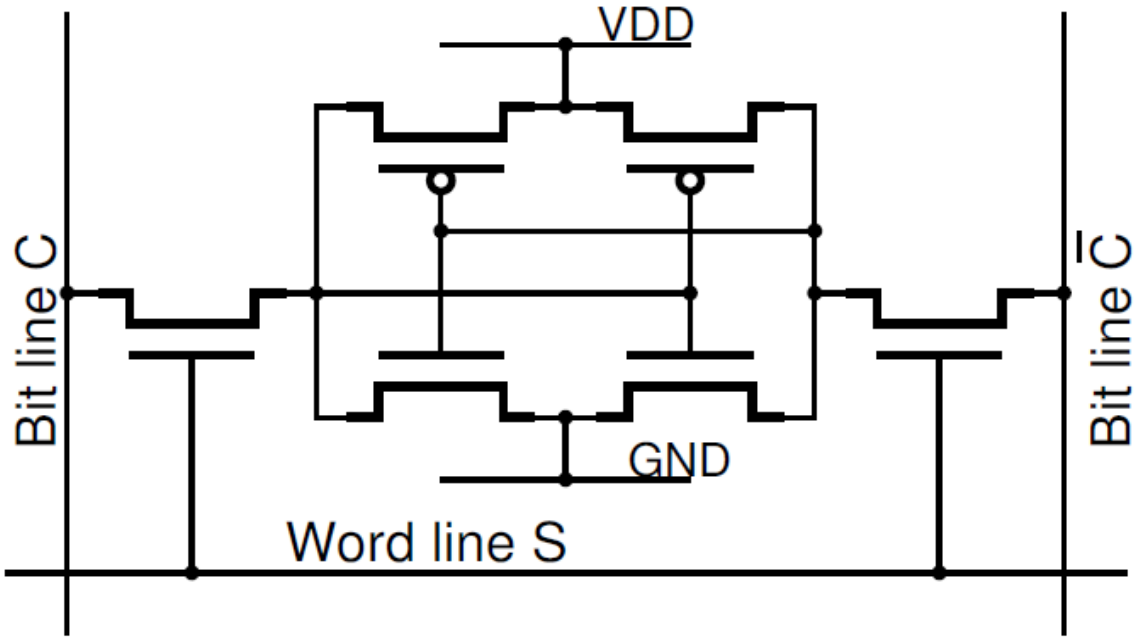


Figure 4.37: A schematic of a CMOS static memory cell

4.15.1 Principles of operations

In order to consider operation of the static read/write memory we have to take into account:

- Relatively large parasitic column capacitances, C_C and $C_{\bar{C}}$
- column pull-up pMOS transistors, as shown in Figure 4.38

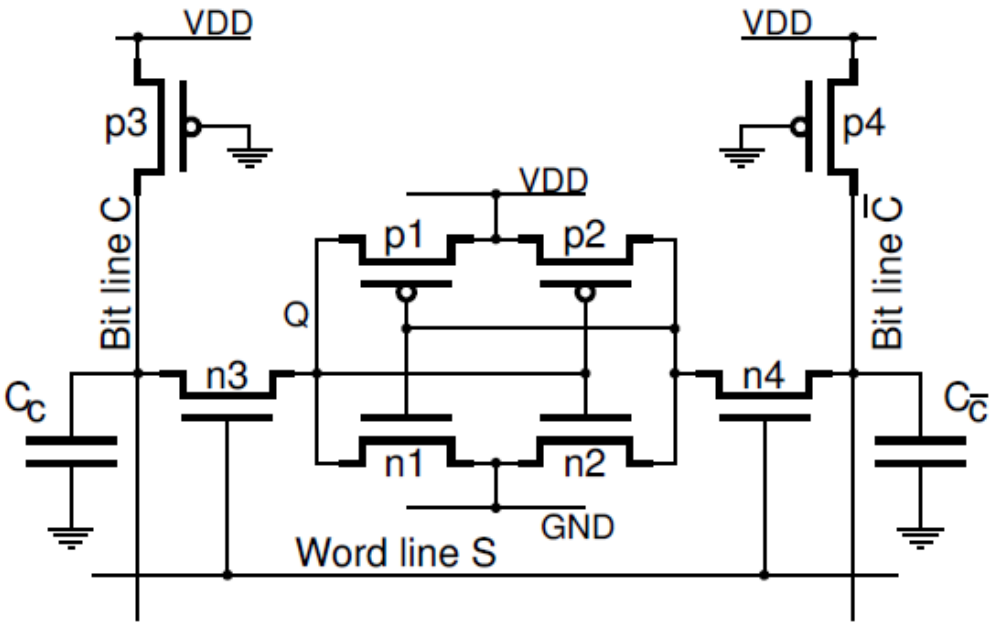


Figure 4.38: A CMOS static memory cell with column pull-up transistors and parasitic column capacitances

When none of the word lines is selected, that is, all S signals are '0', the pass transistors n3, n4 are turned off and the data is retained in all memory cells. The column capacitances are charged by the drain currents of the pull-up pMOS transistors, p3, p4. The column voltages V_C and V c^- both reach the level just below $V_{DD} - V_{Tp}$, say 3.5V for V_{DD} = 5V and the threshold voltage V_{Tp} = 1V.

For the read or write operations we select the cell asserting the word line signal S='1'.For the write operation we apply a low voltage to one of the bit line, holding the other one high. To write '0' in the cell, the column voltage V_C is forced to low(C = 0). This low voltage acts through a related pass transistor (n3) on the gates of the corresponding inverter (n2, p2) so that its input goes high. This sets the signal at the other inverter Q = 0.

Similarly, to write '1' in the cell, the opposite column voltage V_{C^-} is forced to low (C^- = 0) which sets the signal Q = 1.During the read '1' operation, when the stored bit is Q = 1, transistors n3, p1 and n4, n2 are turned on. This maintains the column voltage VC at its steady-state high level (say 3.5V) while the opposite column voltage VC^- is being pulled down discharging the column capacitance CC^- through transistors n4, n2 so that $V_C > V_{C^-}$. Similarly, during the read '0' operation we have $V_C < V_{C^-}$. The difference between the column voltages is small, say 0.5V, and must be detected by the sense amplifiers from data-read circuitry.

4.15.2 SRAM Write Circuitry

The structure of the write circuitry associated with one column of the memory cells is shown in Figure 4.39.

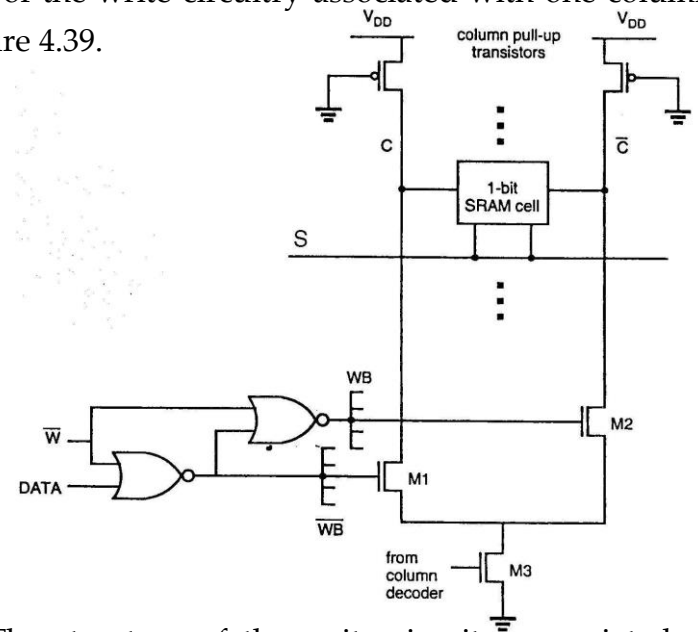


Figure 4.39: The structure of the write circuitry associated with one column of the memory cells.

The principle of the write operation is to assert voltage on one of the columns to a low level. This is achieved by connecting either C or C¯ to the ground through the transistor M3 and either M1 or M2.

The transistor M3 is driven by the signal from the column decoder selecting the specified column. The transistor M1 is on only in the presence of the write enable signal.(W = 0) when the data bit to be written is '0'. The transistor M2 is on only in the presence of the write signal ¯(W = 0) when the data bit to be written is '1'.

4.15.3 SRAM Read Circuitry

The structure of the read circuitry is shown in Figure 4.40.

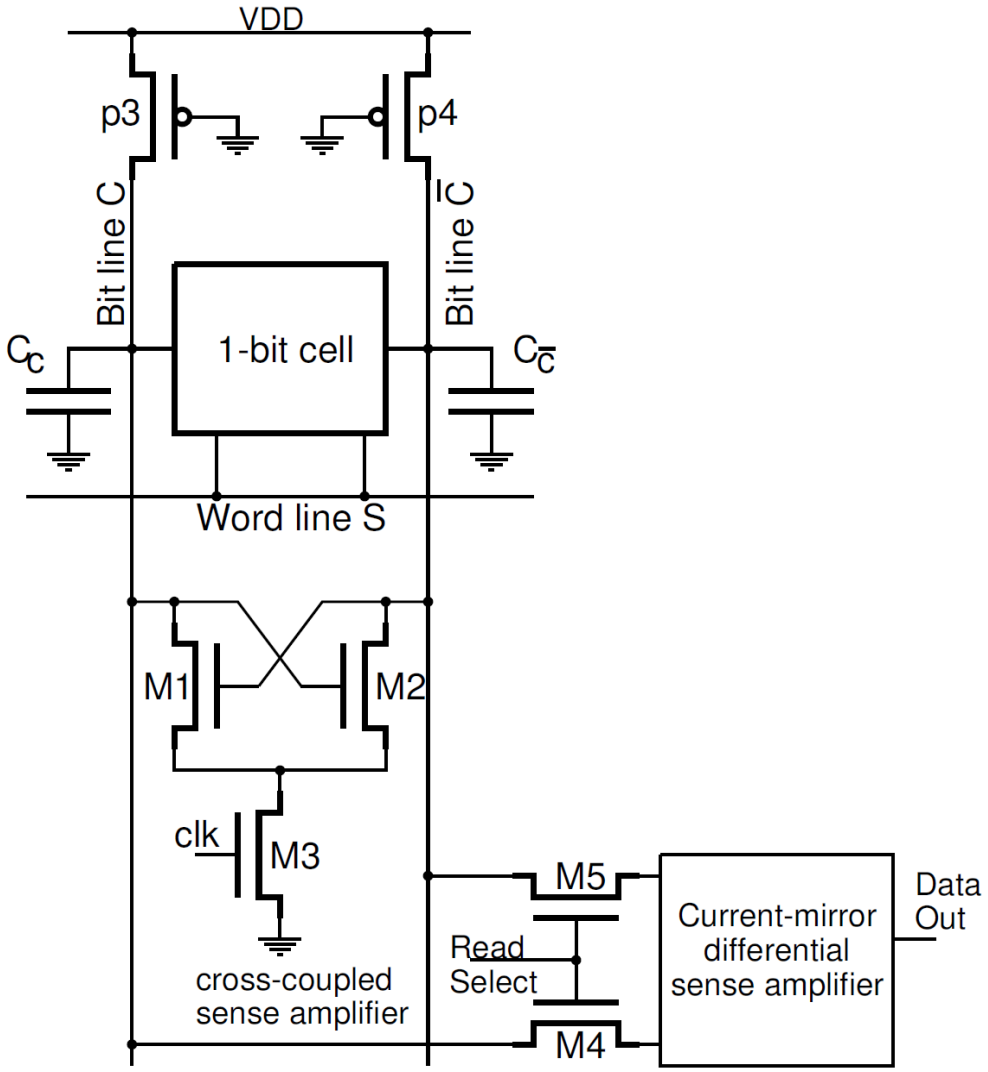


Figure 4.40: The structure of the write circuitry associated with one column of the memory cells.

During the read operation the voltage level on one of the bit lines drops slightly after the pass transistors in the memory cell are turned on.

The read circuitry must properly sense this small voltage difference and form a proper output bit:

'0? If VC < VC⁻

'1? If VC > VC⁻

The read circuitry consists of two level sense amplifiers:

- One simple cross-coupled sense amplifier per column of memory cells,
- One current-mirror differential sense amplifier per the memory chip.

The cross-coupled sense amplifier works as a latch. Assume that the voltage on the bit line C start to drop slightly when the memory access pass transistors are activated by the word line signal S, and that the clk signal is high so that the transistor M3 is turned on. Now, higher voltage on the gate of M1 transistor than on the gate of M2 starts the latching operation which pulls the VC voltage further down switching the transistor M2 off. As a result the parasitic capacitance, CC is discharged through M1 and M3. In this way a small difference between column voltages is amplified.

The amplified (discriminated) column voltages are passed through transistors M4 and M5 to the main sense amplifier.

The schematic of a typical differential current-mirror sense amplifier is shown in Figure 4.41.

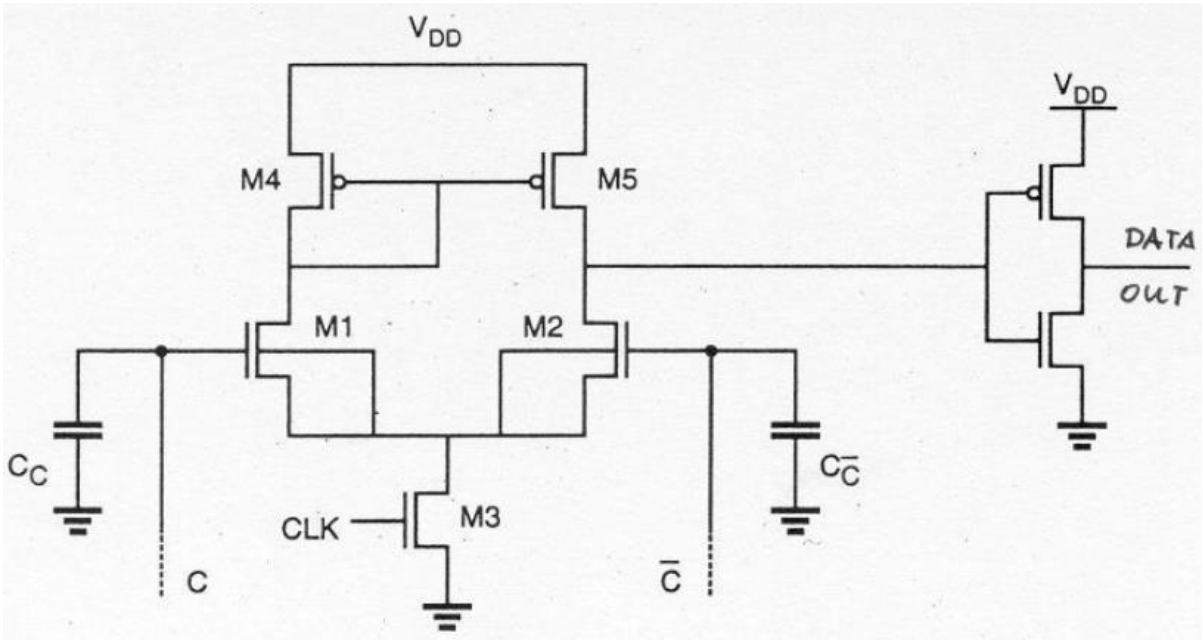
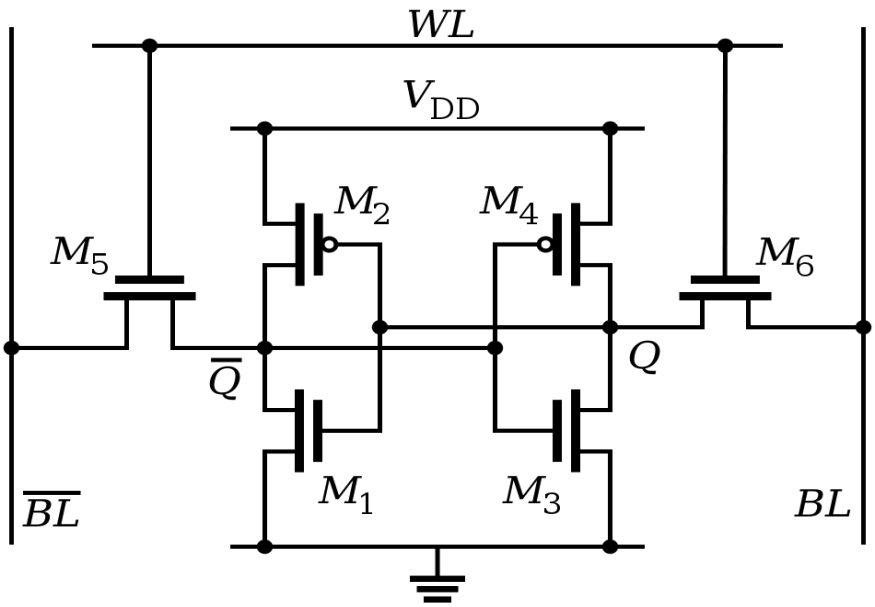


Figure 4.41: A CMOS differential current-mirror sense amplifier.

6-Transistor Cell (Cross Coupled Inverter)

- For larger SRAM modules the above circuit is not very efficient
  - Transistor count per bit is too high



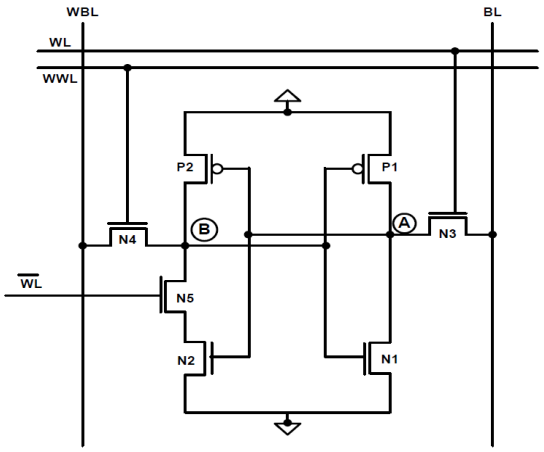
TO READ:

- BIT lines are charged high
- Enable line WL is pulled high, switching access transistors M5 and M6 on`
- If value stored in /Q is 0, value is accessed through access transistor M5 on /BL.
- If value stored in Q is 1, charged value of Bit line BL is pulled up to V_{DD}.
- Value is 'sensed' on BL and /BL.

TO WRITE:

- Apply value to be stored to Bit lines BL and /BL
- Enable line WL is triggered and input value is latched into storage cell
- BIT line drivers must be stronger than SRAM transistor cell to override previous values

While Enable line is held low, the inverters retain the previous value Could use tri-state WE line on BIT to drive into specific state. Transistor count per bit is only 6 + (line drivers & sense logic).



Seven Transistor Memory Cell

4.16 Dynamic Read-Write Memory (DRAM)

In the static CMOS read-write memory data is stored in six-transistor cells. Such a memory is fast and consumed small amount of static power. The only problem is that a SRAM cell occupies a significant amount of silicon space. This problem is addressed in the dynamic read-write memory (DRAM).

In a dynamic RAM binary data is stored as charge in a capacitor. The memory cell consists of a storage capacitor and an access transistor as shown in Figure 4.42.

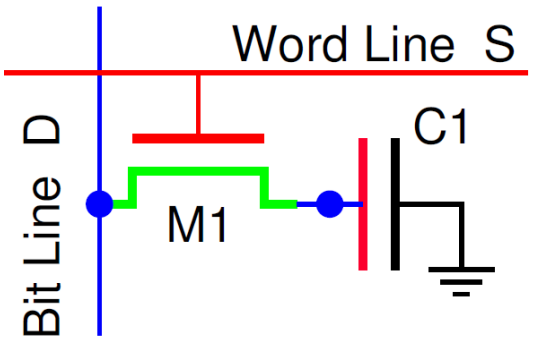


Figure 4.42: A one-transistor DRAM memory cell

Data stored as charge in a capacitor can be retained only for a limited time due to the leakage current which eventually removes or modifies the charge. Therefore, all dynamic memory cells require a periodic refreshing of the stored data before unwanted stored charge modifications occur. Typical storage capacitance has a value of 20 to 50 fF. Assuming that the voltage on the fully charged storage capacitor is V = 2.5V, and that the leakage current is I = 40pA, then the time to discharge the capacitor C = 20fF to the half of the initial voltage can be estimated as

$$t = \frac{1}{2}\frac{C \cdot V}{I} = \frac{20 \cdot 10^{-15} \cdot 2.5}{40 \cdot 10^{-12}} = 0.625\text{ms}$$

Hence ever memory cell must be refreshed approximately every half millisecond.

Despite of the need for additional refreshing circuitry SRAM has two fundamental features which have determined is enormous popularity:

- The DRAM cell occupies much smaller silicon area than the SRAM cell. The size of a DRAM cell is in the order of $8F^2$, where F is the smallest feature size in a given technology. For F = 0.2μm the size is $0.32\mu m^2$
- No static power is dissipated for storing charge in a capacitance. The storage capacitance C_S, which is connected between the drain of the access transistor (the storage node) and the ground, is formed as a trench or stacked Capacitor.

The stacked capacitor is created between a second polysilicon layer and a metal plate covering the whole array area. The plate is effectively connected to the ground terminal.To consider read/write operations we have to take into account a significant parasitic capacitance CC associated with each column, as shown in Figure 4.43.

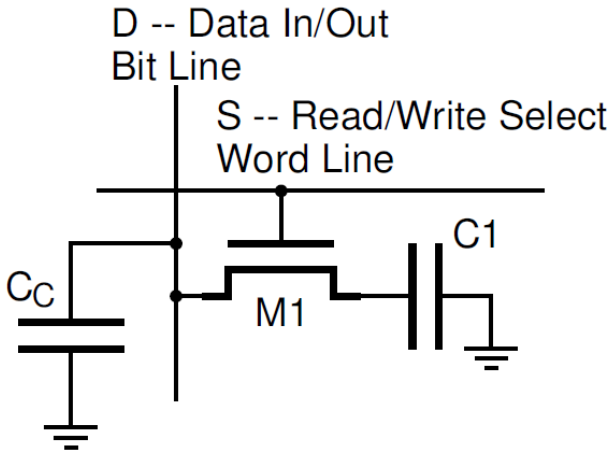


Figure 4.43: A single SRAM cells with a column capacitance shown.

Typically, before any operation is performed each column capacitance b is precharged high.

The cell is selected for a read/write operation by asserting its word line high (S = 1). This connects the storage capacitance to the bit line. The write operation is performed by applying either high or low voltage to the bit line thus charging (write '1') or discharging (write '0') the storage capacitance through the access transistor.

During read operation there is a flow of charges between the storage capacitance C1 and the column capacitance, C_C. As a result the column voltage either increases (read '1') or decreases (read '0') slightly. This difference can then be amplified by the sense amplifier. Note that the read operation destroys the charge stored on the storage capacitance C1 ("destructive readout"). Therefore the data must be restored (refreshed) each time the read operation is performed.

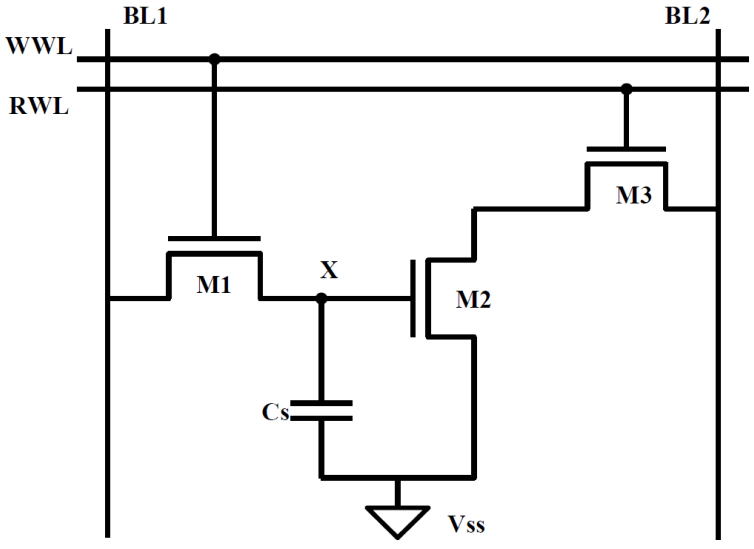
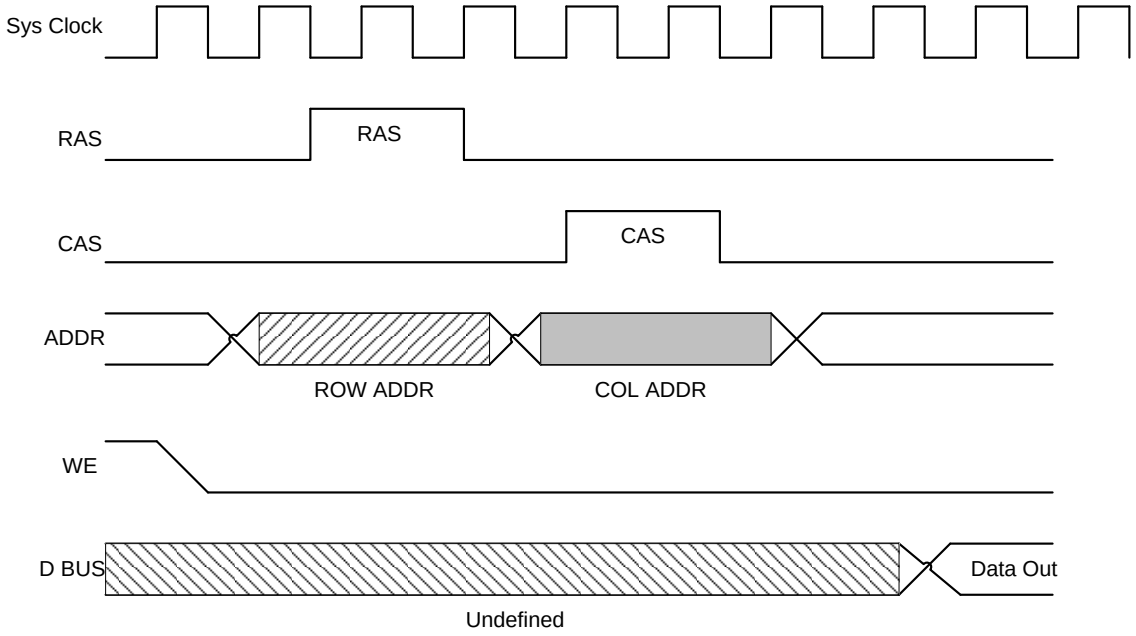


Figure 4.44 Three Transistor Dynamic RAM

The write operation performed is shown for three transistor Dynamic RAM (Figure 1.2) as the appropriate data value is written on BL1 and asserting the write-wordline (WWL). The data is retained as charge on capacitance Cs once WWL is lowered. When reading the cell, the read-wordline (RWL) is raised. The storage transistor M2 is either on or off depending upon the stored value. The bitline BL2 is precharged to VDD before performing read operation. The series connection of M2 and M3 pulls BL2 low when a "1" is stored. BL2 remains high in the opposite case. The cell is inverting; that is, the inverse value of the stored signal is sensed on the bitline.

DRAM Timing:

- DRAM module is asynchronous
  - Timing depends on how long it takes to respond to each operation.



DRAM cannot be read as fast (or as easy) as SRAM

4.17 Serial Access Memories (Serial Memories):

Unlike RAMs which are randomly write the data, serial memories restrict the order of access, which results in either faster access times, smaller area, or a memory with a special functionality.

4.17.1 Shift Registers

Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states. All the flip-flops are driven by a common clock, and all are set or reset simultaneously. There are two types of shift registers; Serial-in-parallel-out and Parallel-in-serial-out.

4.17.2 **Serial-In-Parallel-Out:** In this kind of register, data bits are entered serially. The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously. A construction of a four-bit serial in - parallel out register is shown below.

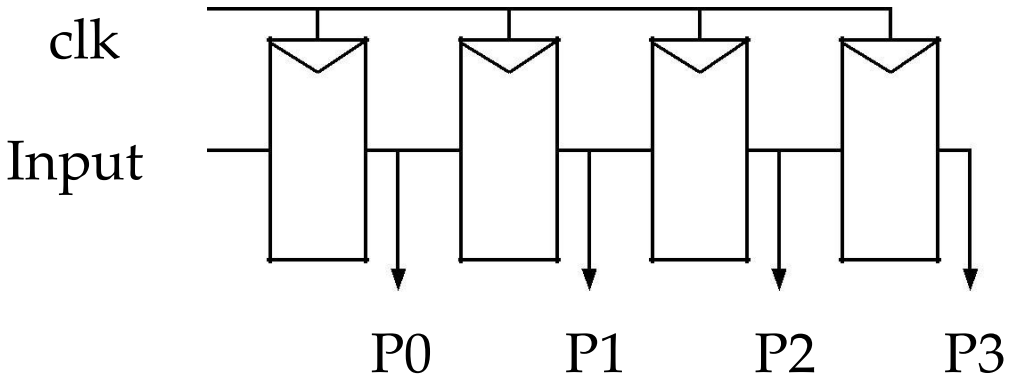


Figure 1.4 Serial-in-parallel-out Shift Register

4.17.3 **Parallel-In-Serial-Out:** The figure shown below is an example of Parallel-In-Serial-Out shift register. P0, P1, P2 and P3 are the parallel inputs to the shift register. When Shift = „0" the shift register loads all the inputs. When Shift = „1" the inputs are shifted to right. This shift register shift one bit per cycle.

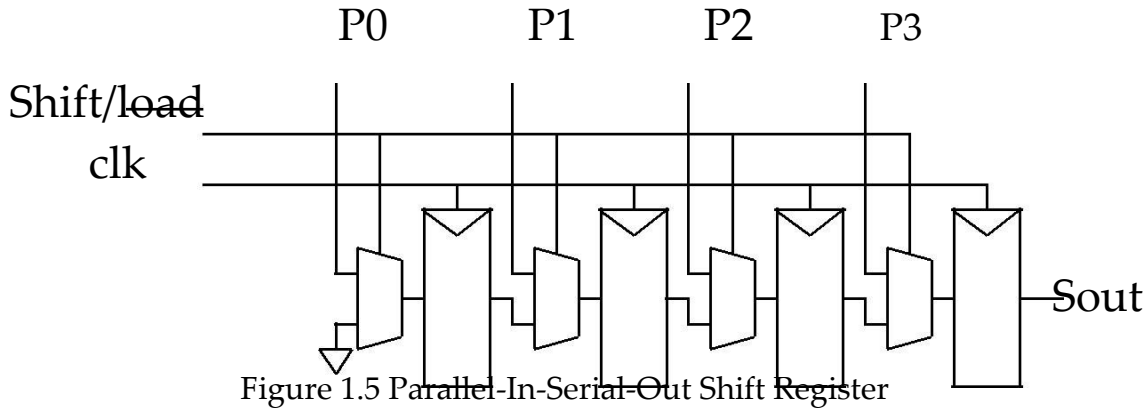


Figure 1.5 Parallel-In-Serial-Out Shift Register

4.17.4 **Queues:**A queue is a pile in which items are added a one end and removed from the other. In this respect, a queue is like the line of customers waiting to be served by a bank teller. As customers arrive, they join the end of the queue while the teller serves the customer at the head of the queue. The major advantage of queue is that they allow data to be written at different rates. The read and write use their own clock and data. There is an indication in queue when it is full or empty. These kind of queues usually built with SRAM and counters. There are two types of queues they are First-In-First-Out and Last-In First-Out.

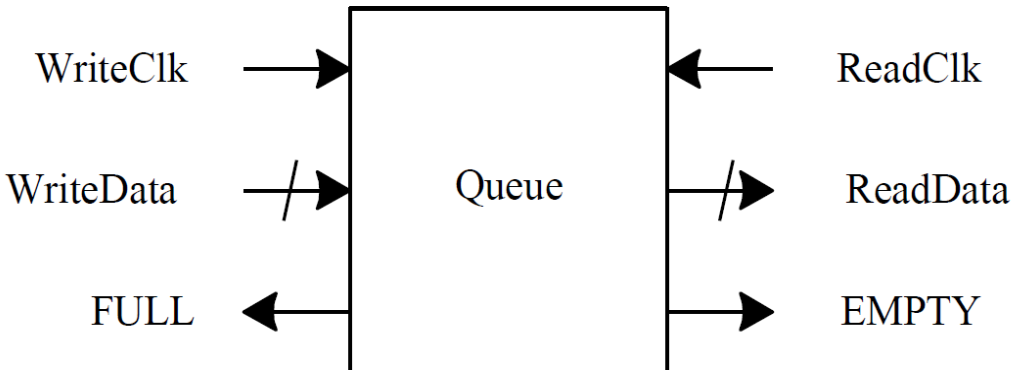


Figure 1.6 Queue

4.17.5 **First-In-First-Out:** In this method initialize the read and write pointers to the first element. Check whether the queue is empty. In write mode we will assign the write pointer and increment the write pointer. If the write almost catches read then queue is full. In read mode we will increment the read pointer.

4.17.6 **Last-In-First-Out:** It is also called as stack; objects which are stored in a stack are kept in a pile. The last item put into the stack is at the top. When an item is pushed into a stack, it is placed at the top of the pile. When an item popped, it is always the top item which is removed. Since it is always the last item to be put into the stack that is the first item to be removed, it is last-in, first-out.

4.17 Contents-Addressable Memory (CAM)

It is another important classification of nonrandom access memories. Instead of using an address to locate a data CAM uses a word of data itself as input in a query-style format. When the input data matches a data word stored in the memory array, a MATCH flag is raised. The MATCH signal remains low if no data stored in the memory corresponds to the input word. This type of memory is also called as associative memory and they are an important component of the cache architecture of many microprocessors.

The Figure 1.7 is an example of 512-word CAM architecture. It supports three modes of operation read, write and match. The read and write modes access and manipulate the data same as in an ordinary memory. The match mode is a special function of associative memory. The data patterns are stored in the comparand block which are needed to match and the mask word indicated which bits are significant. Every row that matches the pattern is passed to the validity block.

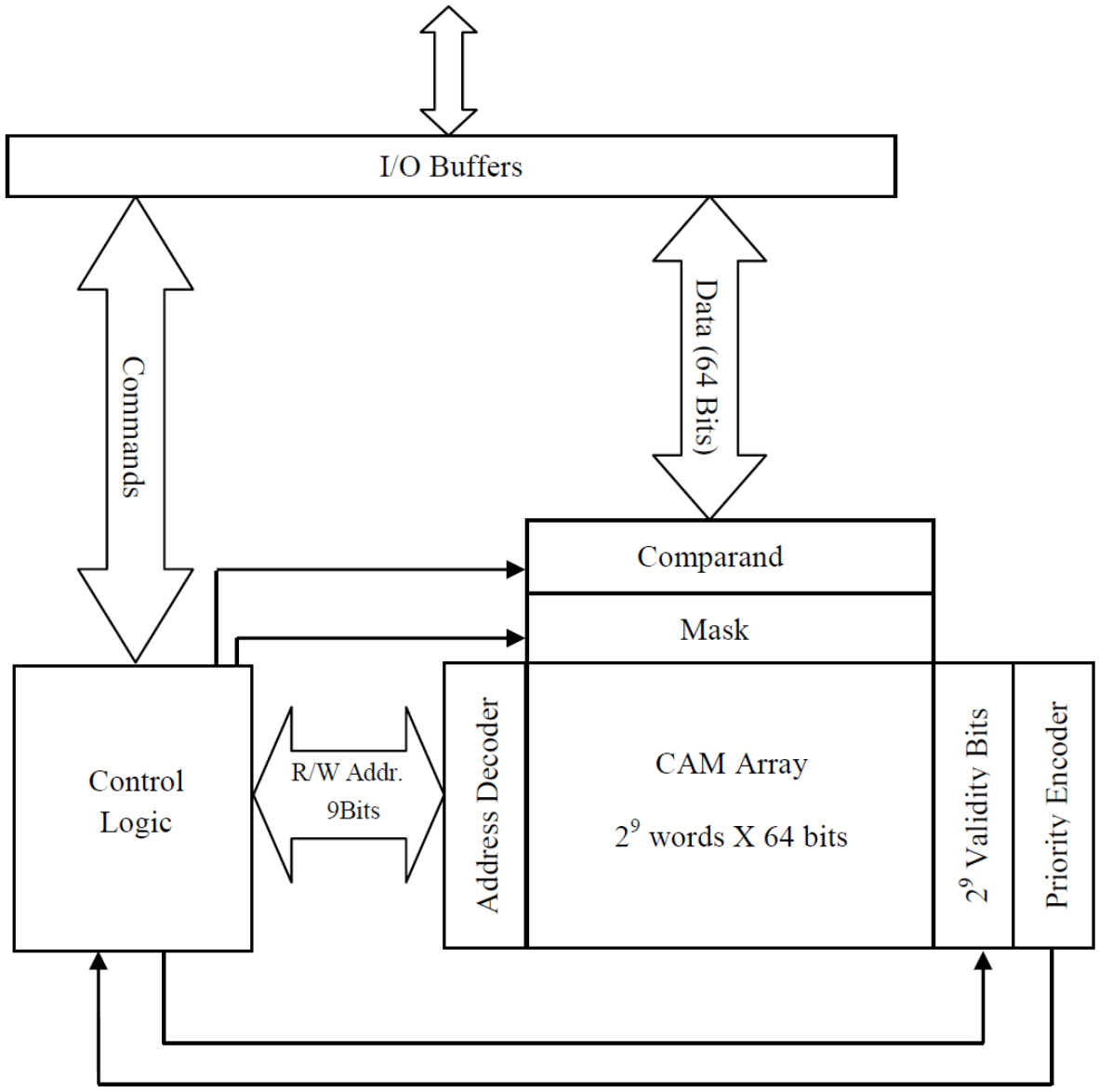


Figure 1.7 Architecture of 512-word CAM

The valid rows that match are passed to the priority encoder leaving the rows that contain invalid data. In the event that two or more rows match the pattern, the address of the row in the CAM is used to break the tie. In order to do that priority encoder considers all the 512 match lines from the CAM array, selects the one with the highest address, and encodes it in binary. Since there are 512 rows in CAM array, it needs 9 bits to indicate the row that matched. There is a possibility that none of the rows matches the pattern so there is one additional „match found" bit provided.