Unit – IV

Low Power Circuit Techniques & SRAM architectures

Minimizing the power consumption of circuits is important for a wide variety of applications, because of the increasing levels of integration and the desire for portability, also it is important to maximize the speed. Low power dissipation and high speed can be accomplished by selecting the optimum circuit architecture.

Arithmetic circuits for most applications is maximizing the speed or throughput. For a growing number of applications, minimizing the power consumption is also of great importance.

The most direct way to reduce the power is to use CMOS circuits, which generally dissipate less power than their bipolar counterparts.

There are four factors which influence the power dissipation of CMOS circuits:

Technology, Circuit design style, Architecture, and Algorithm.

The effect of circuit design style on the power dissipation and delay, the effect of different types of adders in static CMOS circuits, and the effect of architecture on the power dissipation and delay of a multiplier will be discussed.

1. Circuit Design Style

The full adder is the basis for almost every arithmetic unit. There are number of static and dynamic CMOS circuit design styles.

Here seven different full adders, each one designed using a different circuit design style for use in low-power, high-performance arithmetic units will be discussed.

Full Adder Designs

Full adder designed using static complementary MOS logic employing both P- and N-type logic trees is shown in Figure1. The P-logic tree allows the output to be charged high, while the N-tree allows the output to be discharged to ground. Both complemented and uncomplemented inputs are required, and both the true sum and carry and the complemented sum and carry are produced. The sum and carry functions are computed independently of each other.

This full adder has 30 transistor count.

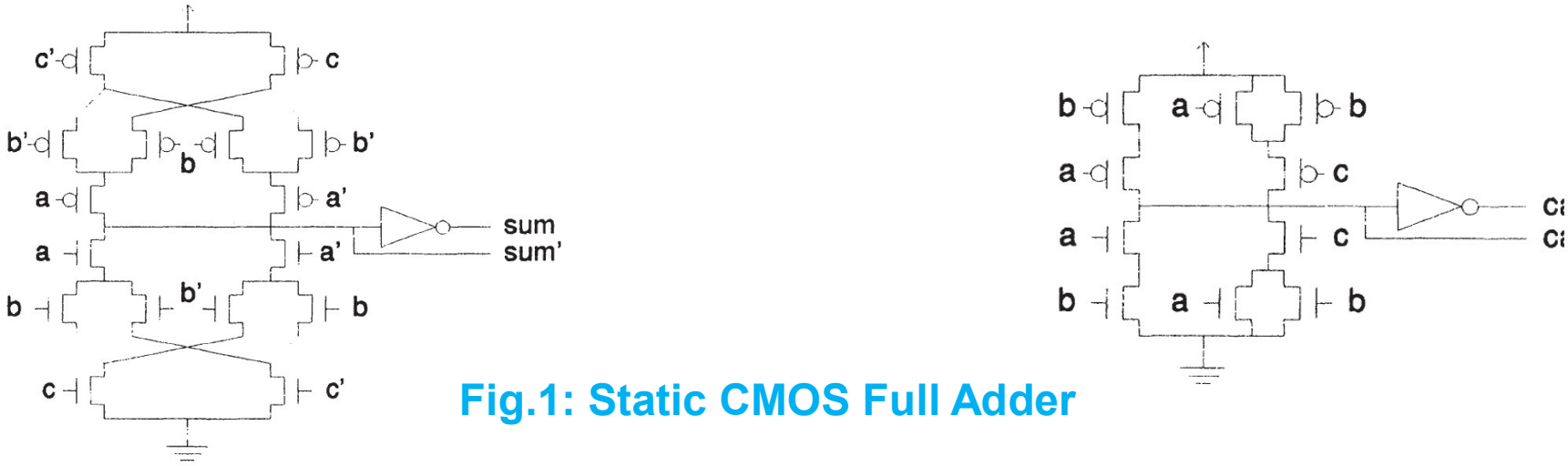


Fig.1: Static CMOS Full Adder

NORA logic is unique among the CMOS families, it does not require complemented inputs and does not compute both complemented and uncomplemented outputs.

Full adders constructed using NORA dynamic CMOS logic (NORA) employ alternating stages of P- and N-type logic trees to form the carry and sum outputs as shown in Figure2. The P-type stage that forms the carry output is dynamically precharged high, while the N-type transistor tree that computes the sum output is dynamically pre-discharged low. This precharging and pre-discharging process requires a two-phase complimentary clock, denoted as phi and phi'. It requires 22 transistors.

$$C_{out} = AB + C(A + B)$$

$$S = ABC + (A + B + C)\overline{C}_{out}$$

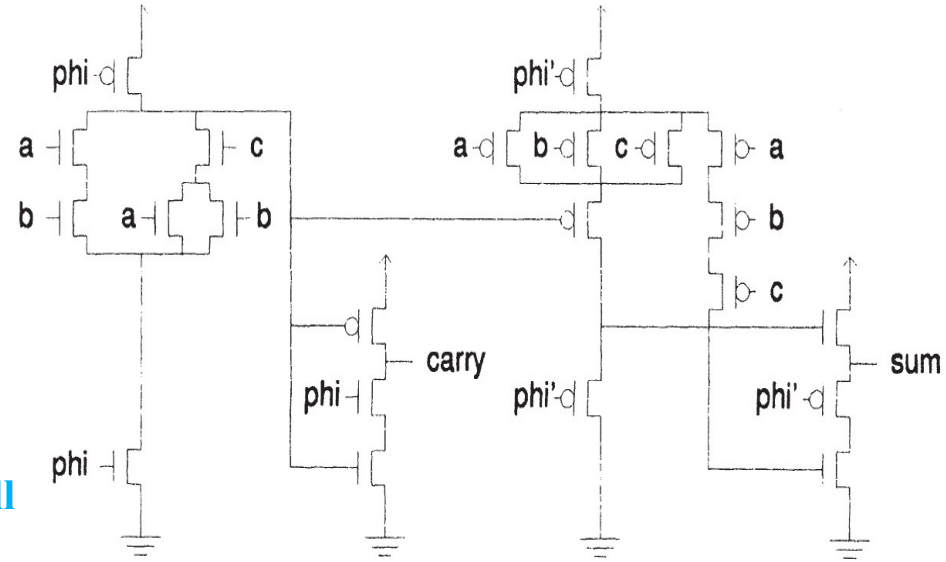
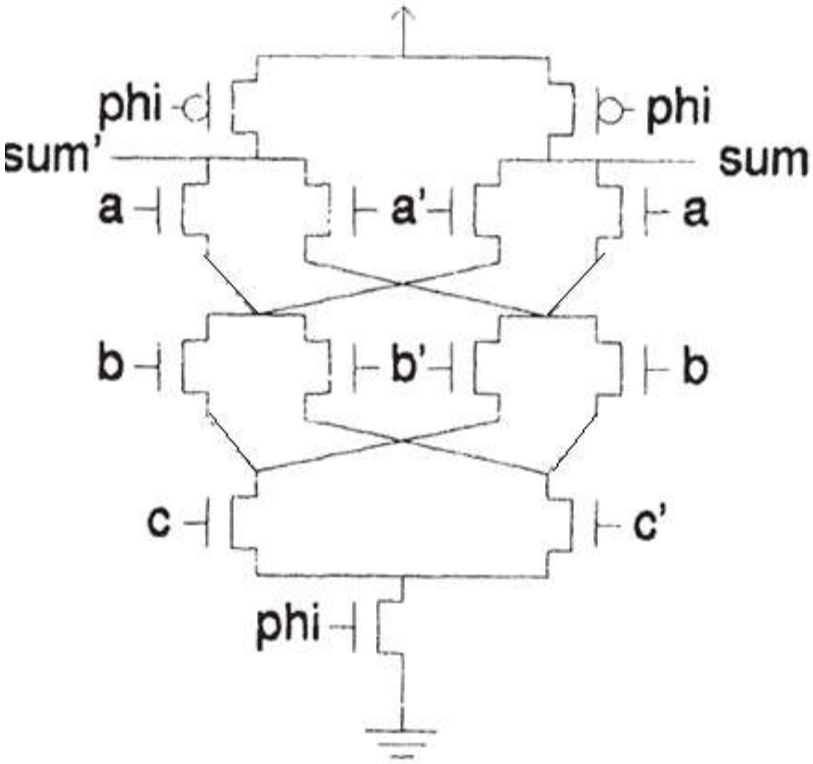


Fig.2:NORAce dynamic CMOS logic (NORA) Full Adder

Cascode Voltage Switch Logic (CVSL) is a dynamic logic family. It requires single phase clock. In the CVSL full adder circuit shown in Figure3, the outputs and their complements are all precharged high when the clock is low. When the clock signal goes high, the complementary cascoded differential N-type transistor trees pull either the output or its complement low. Again, the sum and carry are computed independently of each other.

A	B	C	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



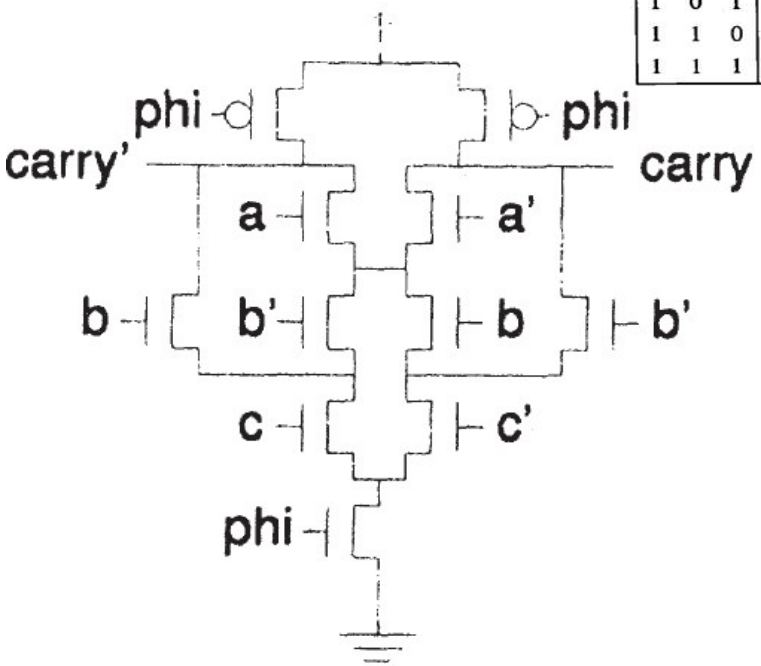


Fig.3: Cascode Voltage Switch Logic (CVSL) Full Adder

Replacing the P-type transistors in CVSL with a cross coupled pair of P transistor yields a Static Differential Cascode Voltage Switch (DCVS) logic as shown in Figure4. The cross coupled P transistors act as a differential pair. When the output at one side gets pulled low, then the opposite P transistor will be turned on, and the output on that side will be pulled high.

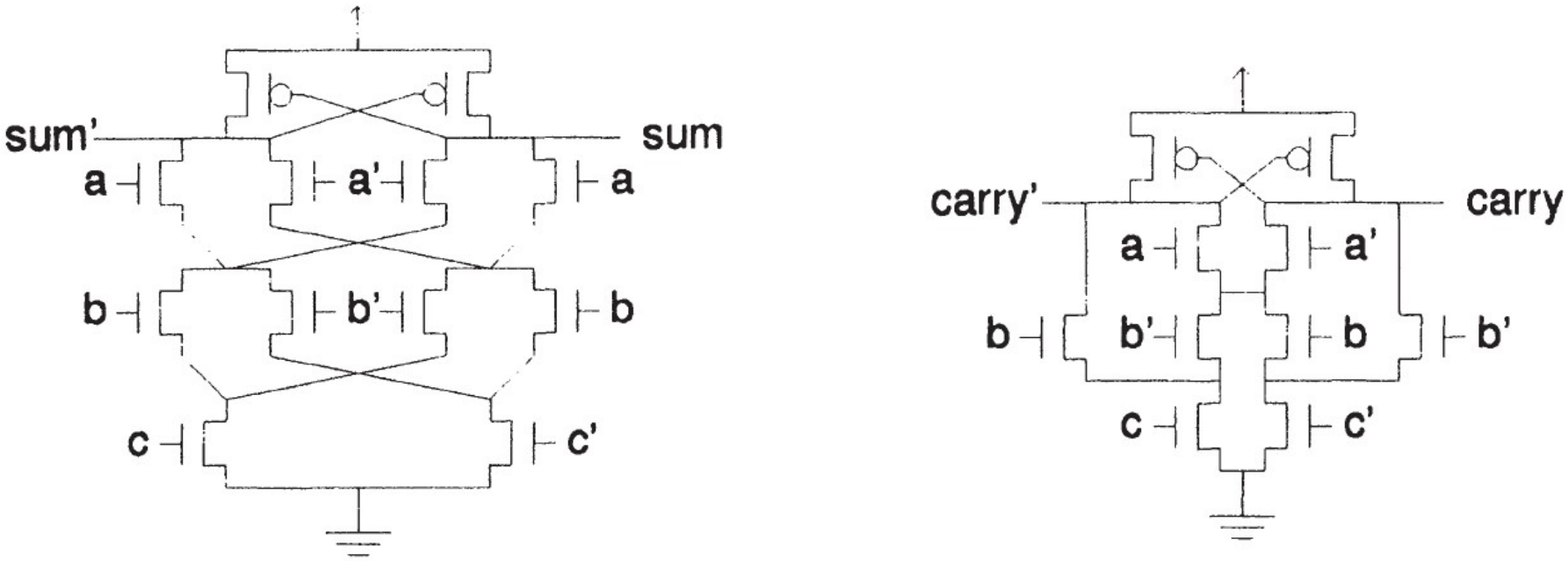


Fig.4: Differential Cascode Voltage Switch Logic (DCVSL) Full Adder

CMOS NonThreshold Logic (CNTL) employs the same binary decision trees and cross coupled P transistors as DCVS, but extra N-type transistors are added as shown in Figure5, between the cross coupled P transistors and the output in order to lower the voltage level of the high output.

Another two N transistors are placed between the N transistor trees and ground in order to raise the voltage level of the low output. Shunt capacitors are also placed across these last two transistors in order to reduce negative feedback. The resulting reduced voltage swing allows the CNTL full adder to switch more quickly than the DCVS full adder. This full adder requires 30 transistors and 4 capacitors.

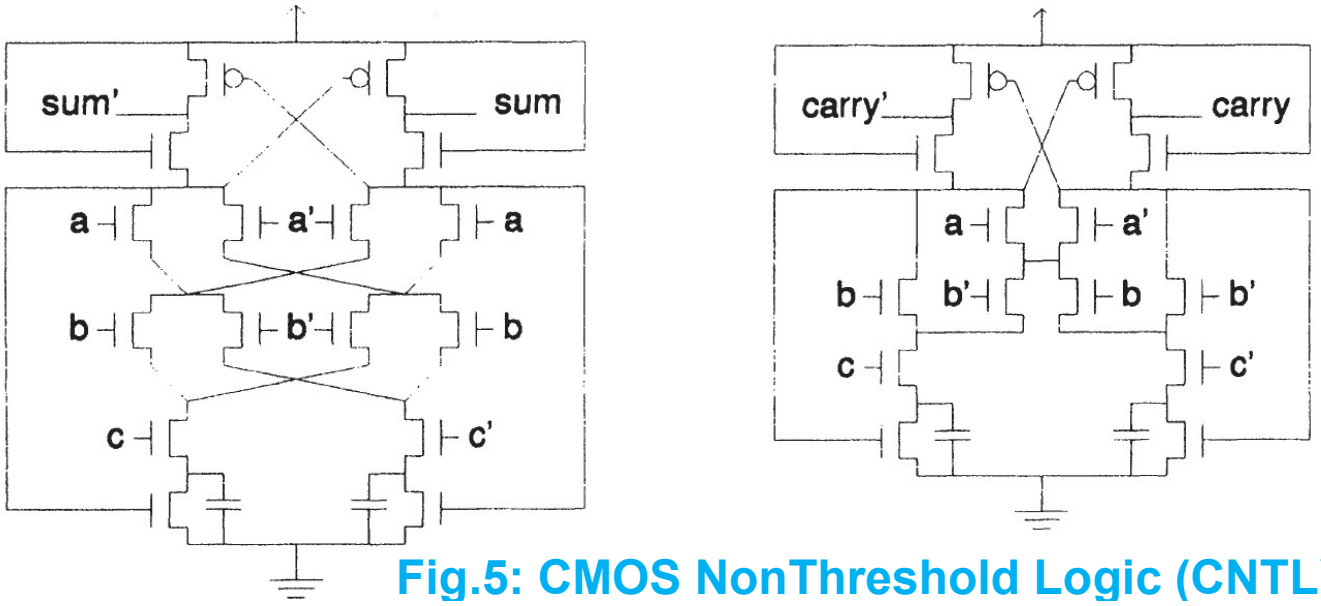


Fig.5: CMOS NonThreshold Logic (CNTL) Full Adder

Another extension to the DCVS full adder is the addition of self-timing signals, as shown in Figure6. Enable/disable CMOS Differential Logic (ECDL) uses a completion signal, DONE, to pre-discharge the outputs of each stage.

While the DONE input signal is held high, the sum and carry outputs are discharged.

The DONE input signal goes low(active low) when the previous stage has finished its computation. Once the DONE input signal goes low, the full adder operates in the same manner as the standard DCVS full adder. This full adder requires 35 transistors.

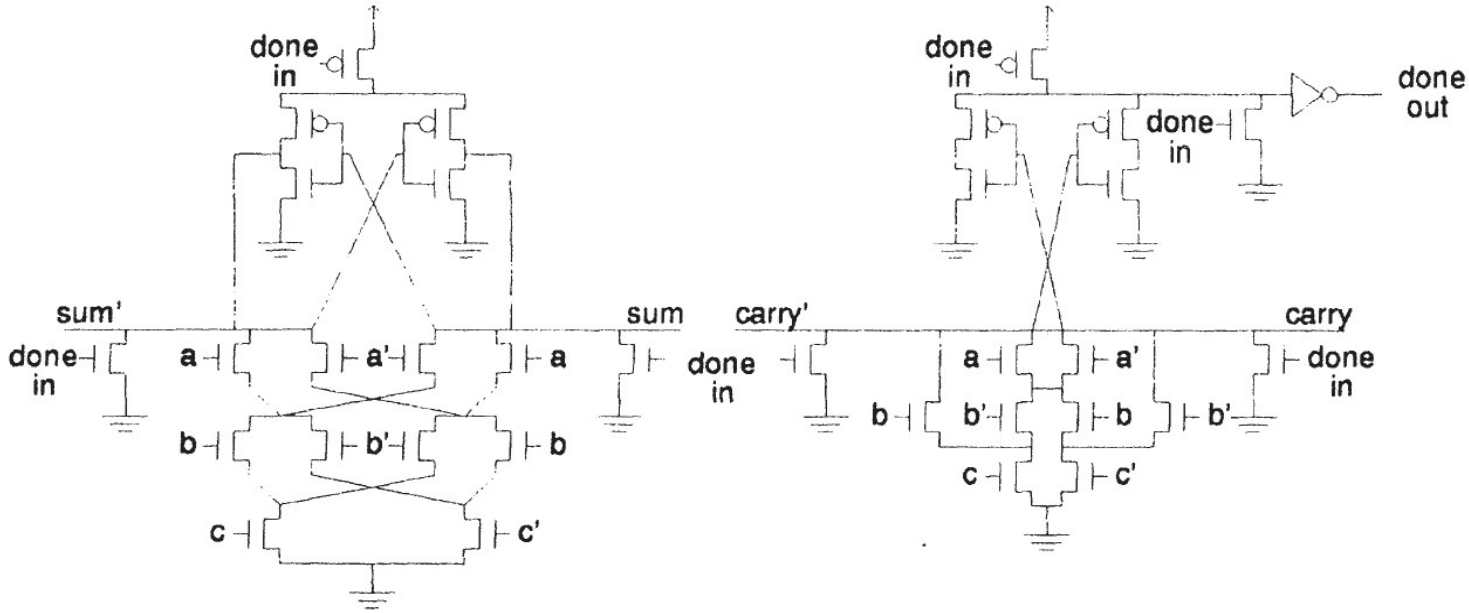


Fig.6: EnableIDisable CMOS Differential Logic (ECDL) Full Adder

Enhancement Source Coupled Logic (ESCL) is another variation on the DCVS full adder. As shown in Figure7, it replaces the P-type transistors with two biased N-type transistors, whose inputs are held at voltage *Vb*. The value for *Vb* is about a diode drop less than *Vdd*. This circuit is further biased by the insertion of a constant current source in between the bottom of the N-type transistor tree and ground. Inputs to the ESCL circuit steer current from one branch of the tree to another. The current source is a simple 2 transistor current mirror, the total transistor count of this full adder to 24 transistors.

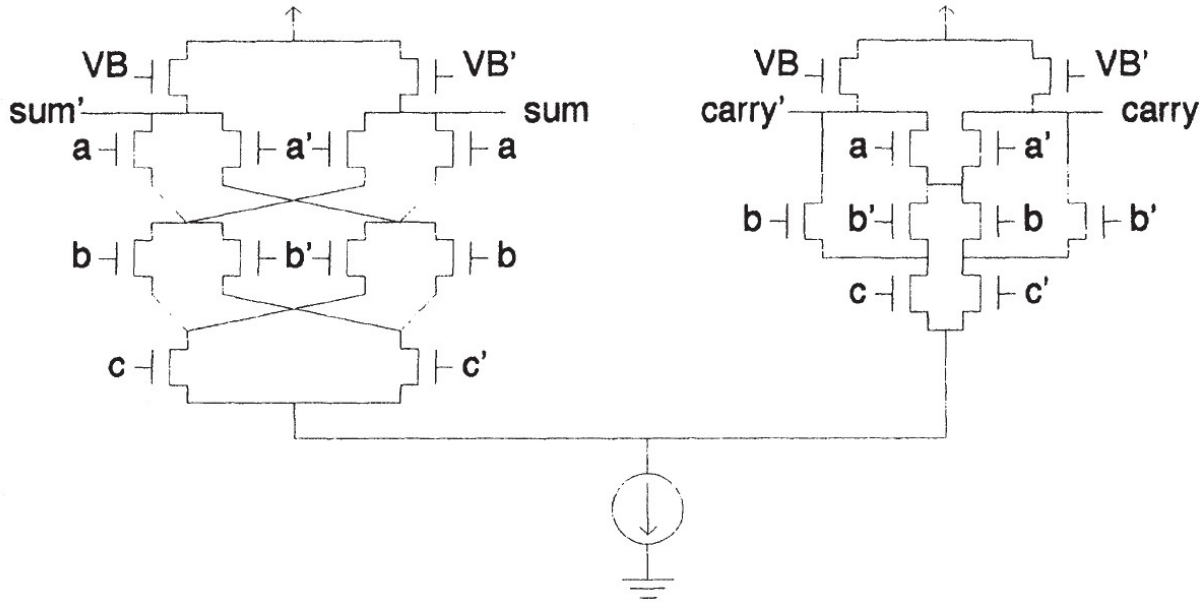


Fig.7: Enhancement Source Coupled Logic (ESCL) Full Adder

Full Adder Transistor Count and Area

	Transistors	Rank	Area (μ^2)	Rank
CMOS	30	5	21,294	3
NORA	22	1	14,319	1
CVSL	24	3	25,740	4
DCVS	22	1	21,080	2
CNTL	34	6	40,020	7
ECDL	35	7	34,170	6
ESCL	24	3	26,522	5

Table x1: Full Adder Transistor Count and Area

One of the primary concerns in VLSI designs is minimizing the area which is often approximated by the number of transistors, as shown in Table x1. The area considered is 2 micron technology. As expected, the adder with the largest and smallest numbers of transistors, occupy the largest and smallest areas, respectively. The NORA adder has an especially low area, likely due to the fact that it does not use either uncomplemented inputs or outputs. This reduces the amount of interconnect required, thus reducing the total area.

2. Adder Types

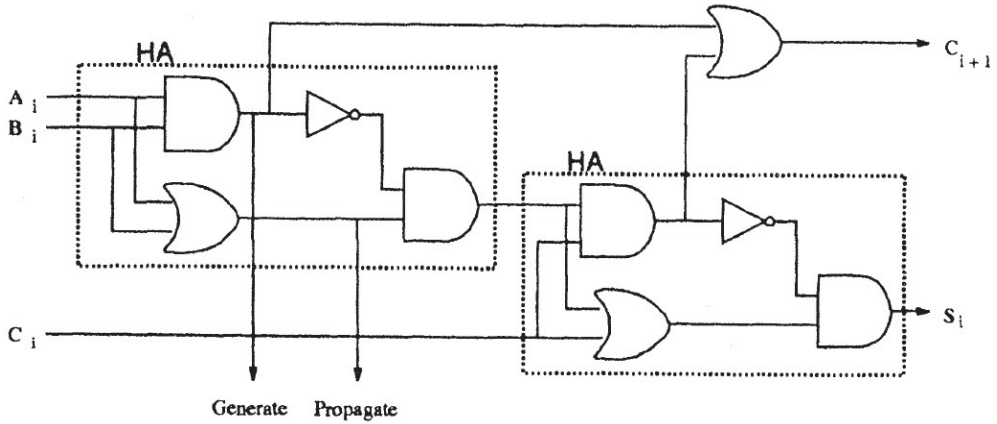
The following types of adders are Analysed:

Ripple Carry, Constant Block Width Single-level Carry Skip, Variable Block Width Multi-level Carry Skip, Carry Lookahead, Carry Select, and Conditional Sum.

The N-bit Ripple Carry adder consists of N full adders (shown in Figure 8) with the carry output of the full adder connected to the carry input of its most significant neighbor as shown in Figure9.

This is the smallest possible parallel adder. Unfortunately, it is also the slowest. Since the adders are connected by the carry chain, a worst case addition will require that a carry ripple from the least significant full adder through intermediate full adders to the most significant full adder.

If all of the carries were known ahead of time, the addition could be completed in only 6 gate delays. That is the amount of time it takes for the two input bits to propagate to the sum output of a single full adder.



$$C_{out} = AB + C(A + B)$$

Fig8: Full Adder

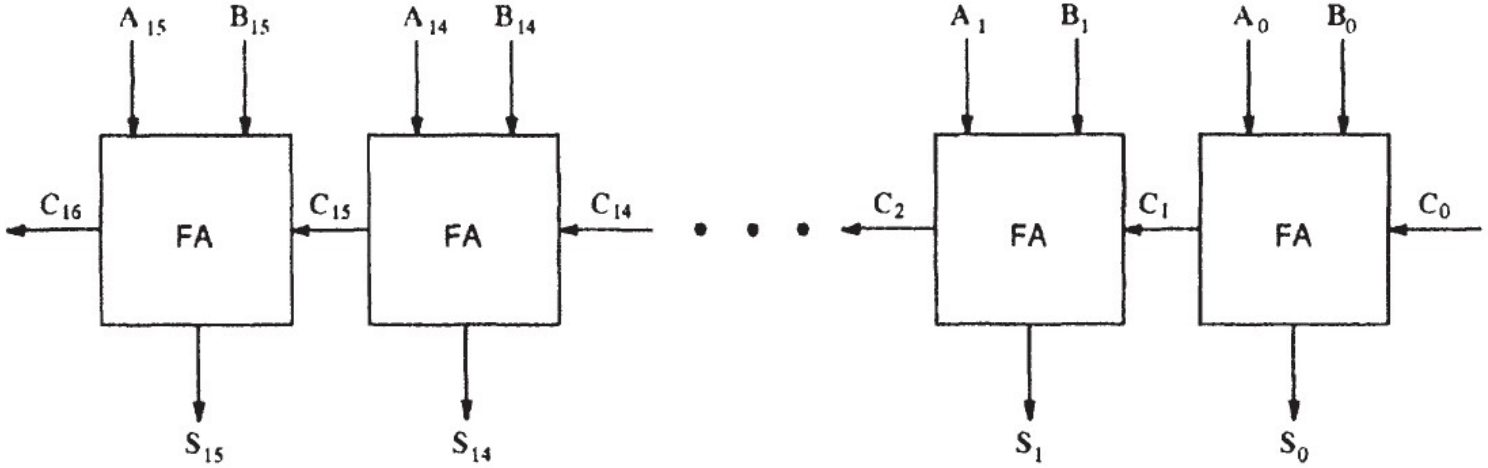


Fig9: 16 Bit Ripple Carry Adder

Since the sum delay is very small compared to the carry delay all of the other adders discussed in this subsection use additional logic to speed up carry generation and/or propagation.

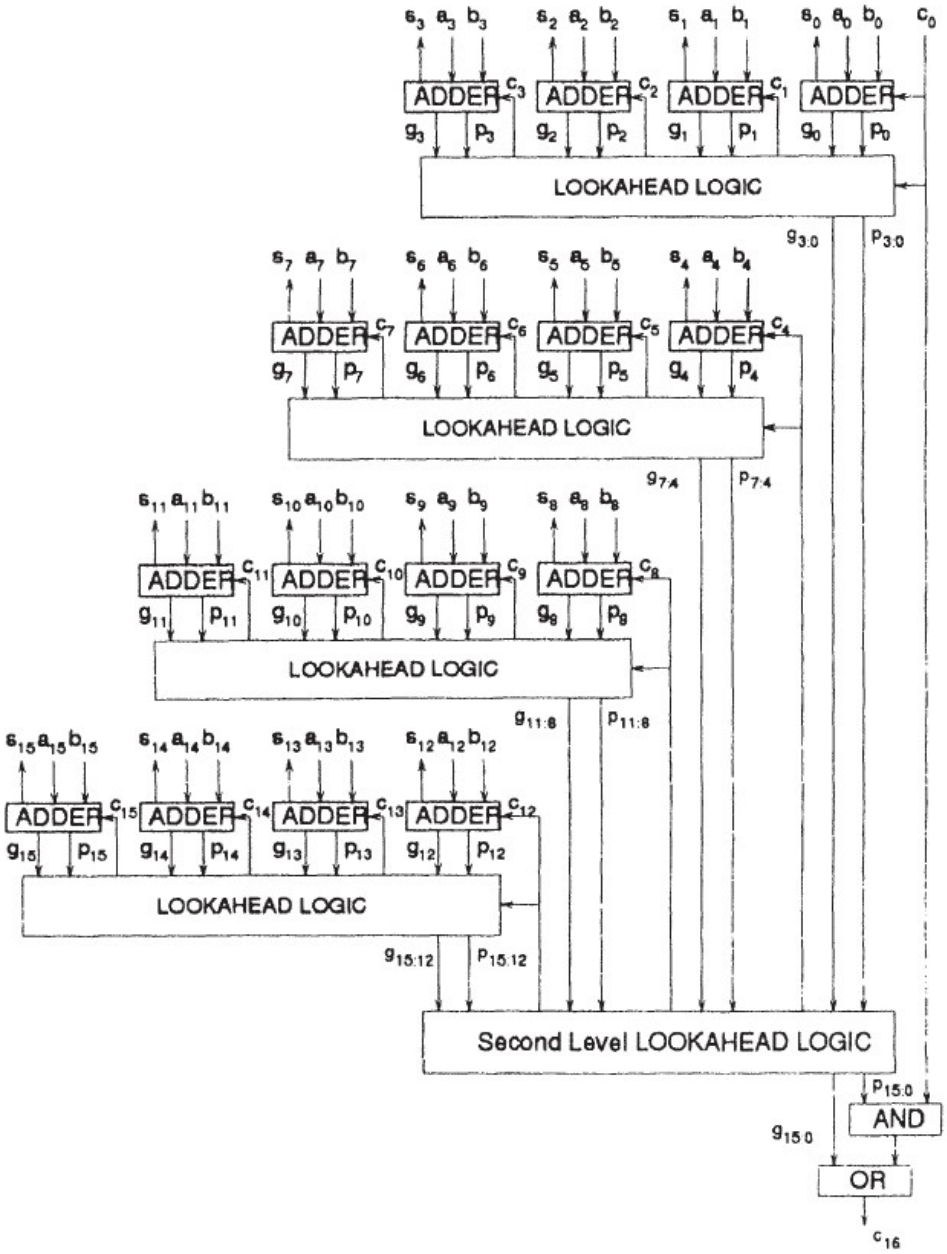


Fig.10: 16-Bit Carry Lookahead Adder

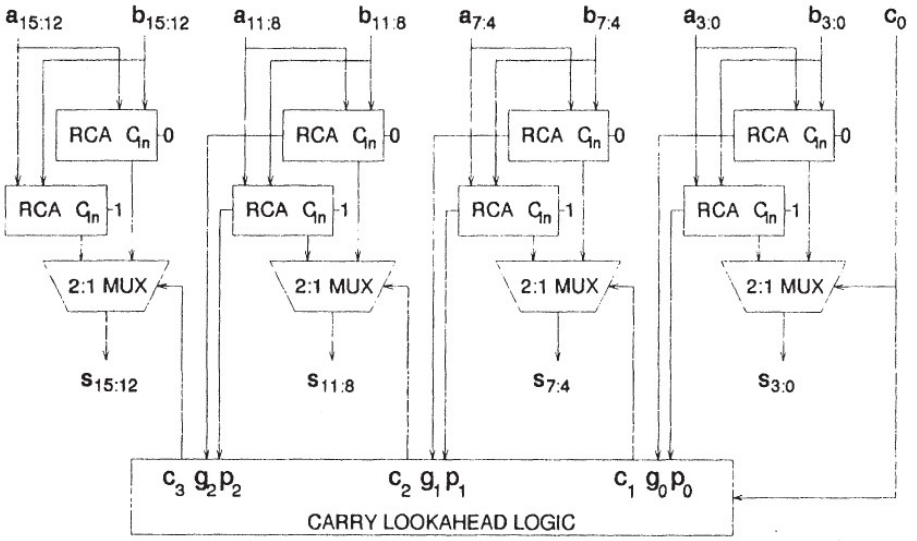


Fig.11: 16-Bit Carry Select Adder (CSA

For the Carry Lookahead adder shown in Figure10, the OR gate used in the full adder to generate the carry out is eliminated, resulting in an adder with only eight gates. The carry input for each full adder is computed by a tree of lookahead logic blocks. Each block receives *pi.* and *gi.* signals from the full adders, and computes the carries. lookahead logic is implemented with four-bit modules, each of which has 14 gates.

The carry select adder consists of pairs of k-bit wide blocks, each block consisting of a pair of ripple carry adders and a multiplexer controlled by the incoming carry signal. The carry generation is done using the same carry lookahead modules used in the carry lookahead adder.

The Carry Skip adder uses the idea that if corresponding bits in the two words to be added are not equal, a carry signal into that bit position will be passed to the next bit position.

In the Constant Block Width Carry Skip adder shown in Figure12, the optimal width of each block is determined based on the number of bits to be added.

This implementation uses only a single level of skip logic. The optimal block size for a 16 bit adder is 3 bits, so that the 16 bit adder is realized by dropping the two most significant bits of an 18 bit adder.

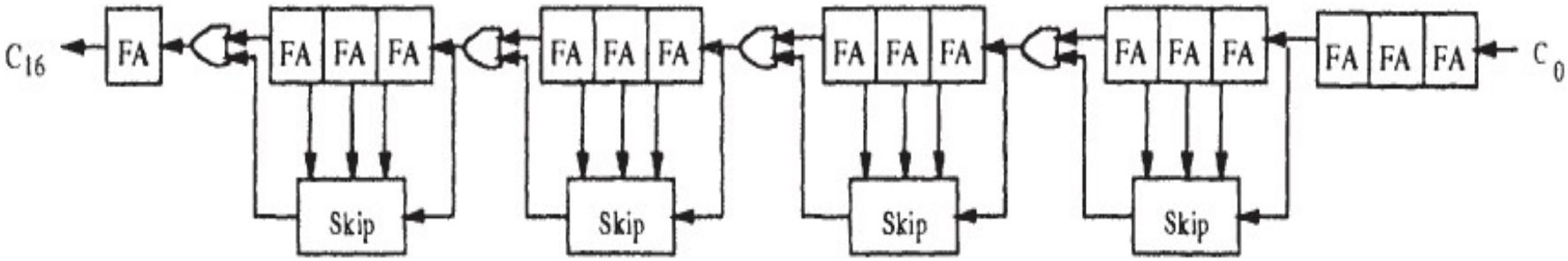


Fig.12: 16 Bit Constant Block Width Carry Skip Adder

Variable Block Width Carry Skip adder shown in Figure13, multiple levels of skip logic and variable block sizes are used in order to create a faster circuit. The Turrini algorithm is used to generate the optimal grouping of skip groups and block sizes. Application of this algorithm generally results in an adder larger than the desired size. The unnecessary full adders and skip logic on the most significant end can be eliminated without affecting the delay.

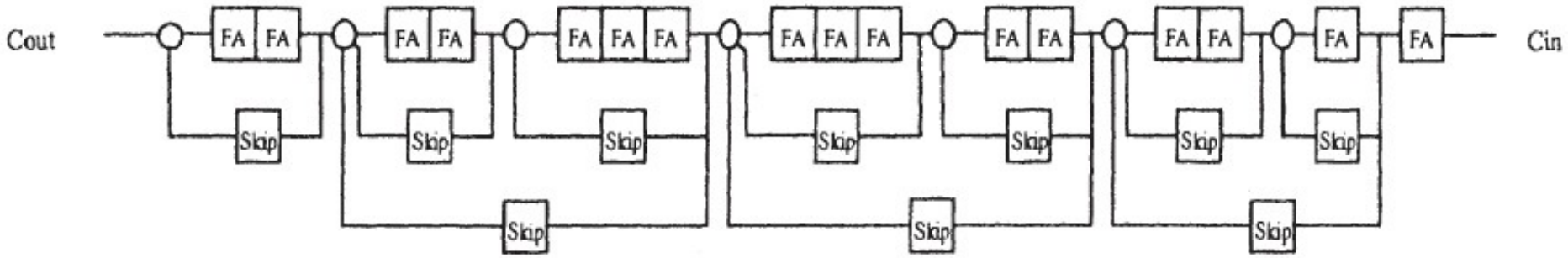


Fig.13: 16 Bit Variable Block Width Carry Skip Adder

The Conditional Sum adder shown in Figure14 uses 2:1 multiplexers to recursively combine successively larger blocks of conditional sum and carry bits.

The basic cell of this adder is based on "H" cell. This circuit accepts as inputs the two bits to be added, and produces four outputs: the sum and carry for cin=0, and the sum and carry for Cin=1.

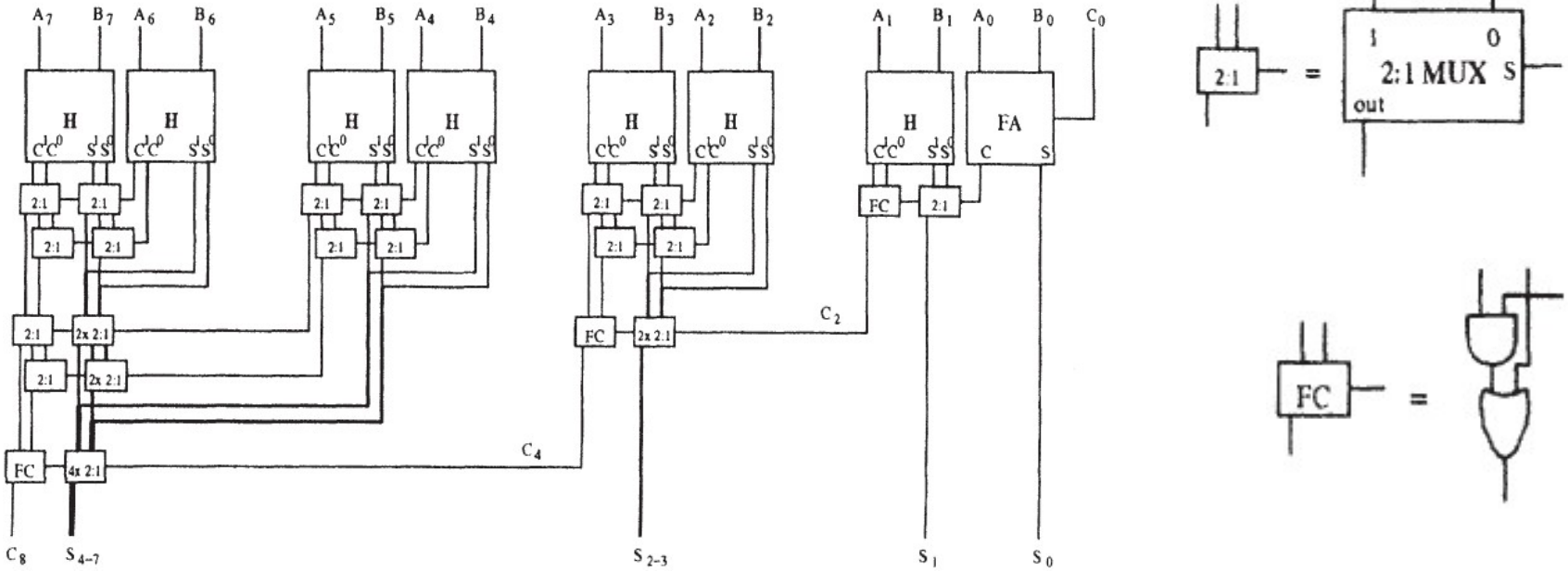


Fig.14: 8-Bit Conditional Sum Adder

In choosing an adder for a particular application, several things must be considered, including speed, size, and dynamic power consumption. The speed is usually considered to be the worst case number of gate delays from input to the slowest output. Table x2 presents the worst case number of gate delays for the six adders by assuming all gates have the same delay, regardless of the fan-in or fan-out and Table x3 gives the size of each adder in terms of number of gates.

Adder Type	Adder Size		
	16	32	64
Ripple Carry	36	68	132
Constant Width Carry Skip	23	33	39
Variable Width Carry Skip	17	19	23
Carry Lookahead	10	14	14
Carry Select	14	14	14
Conditional Sum	12	15	18

Table x2: Worst Case Delay (in gate units)

Adder Type	Adder Size		
	16	32	64
Ripple Carry	144	288	576
Constant Width Carry Skip	156	304	608
Variable Width Carry Skip	170	350	695
Carry Lookahead	200	401	808
Carry Select	284	597	1228
Conditional Sum	368	857	1938

Table x3: Number of Gates

Multiplier Types: i) Array Multiplier: Array Multiplier circuit is based on repeated addition and shifting procedure. Each partial product is generated by the multiplication of the multiplicand with one multiplier digit.

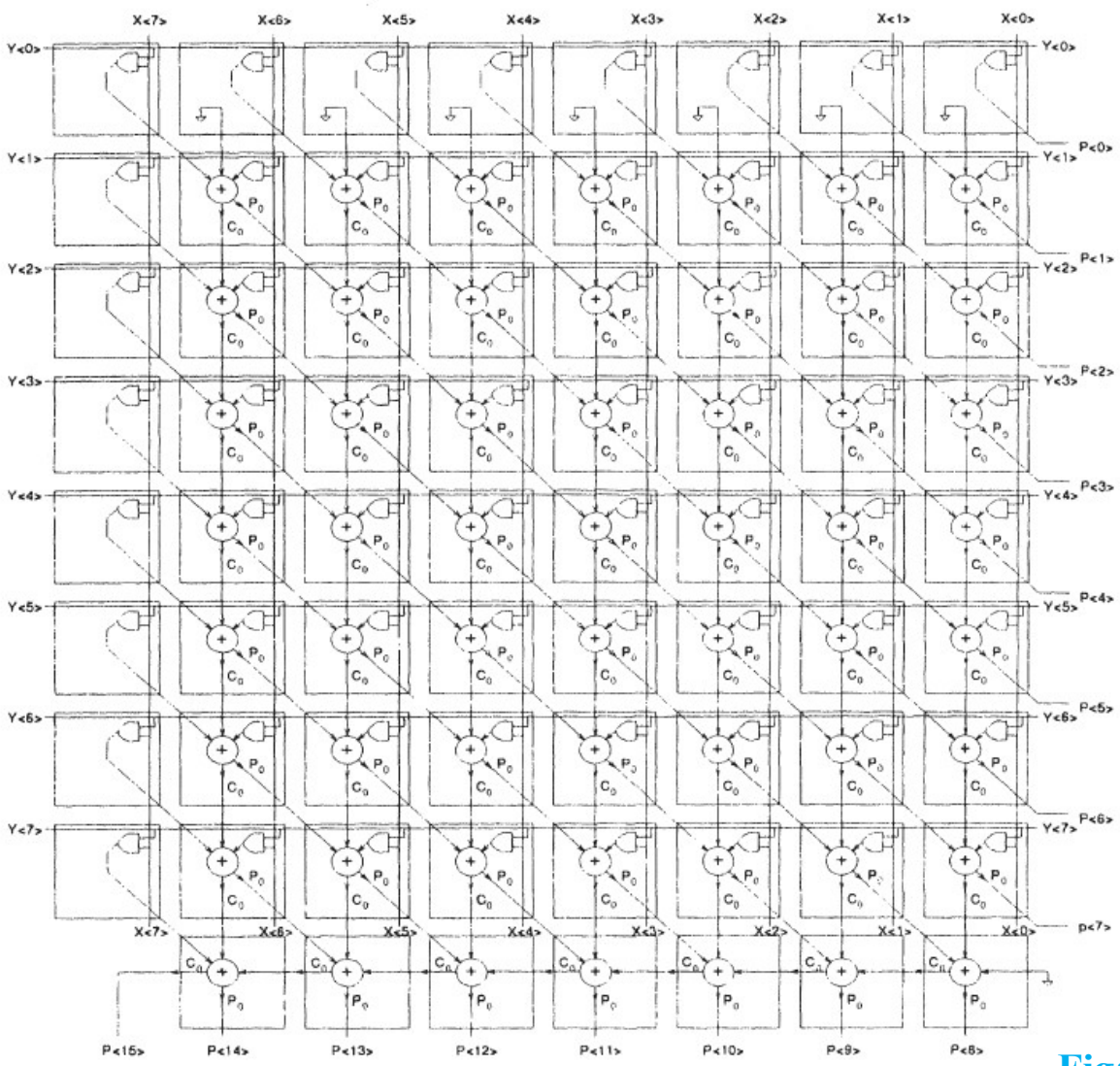


Figure: 8-Bit Array Multiplier

				a_3	a_2	a_1	a_0	
				b_3	b_2	b_1	b_0	
				a_0b_3	a_0b_2	a_0b_1	a_0b_0	
			a_1b_3	a_1b_2	a_1b_1	a_1b_0		
		a_2b_3	a_2b_2	a_2b_1	a_2b_0			
	a_3b_3	a_3b_2	a_3b_1	a_3b_0				
P_7	P_6	P_5	P_4	P_3	P_2	P_1	P_0	
			1	0	1	0	X	1 1 0
			1	0	1	0		
		0	0	0	0			
	1	0	1	0				
1	0	1	0					
1	0	0	0	0	0	1	0	

Illustration of 4-bit Array Multiplication

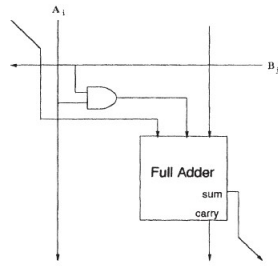


Fig: One Cell of an Array Multiplier

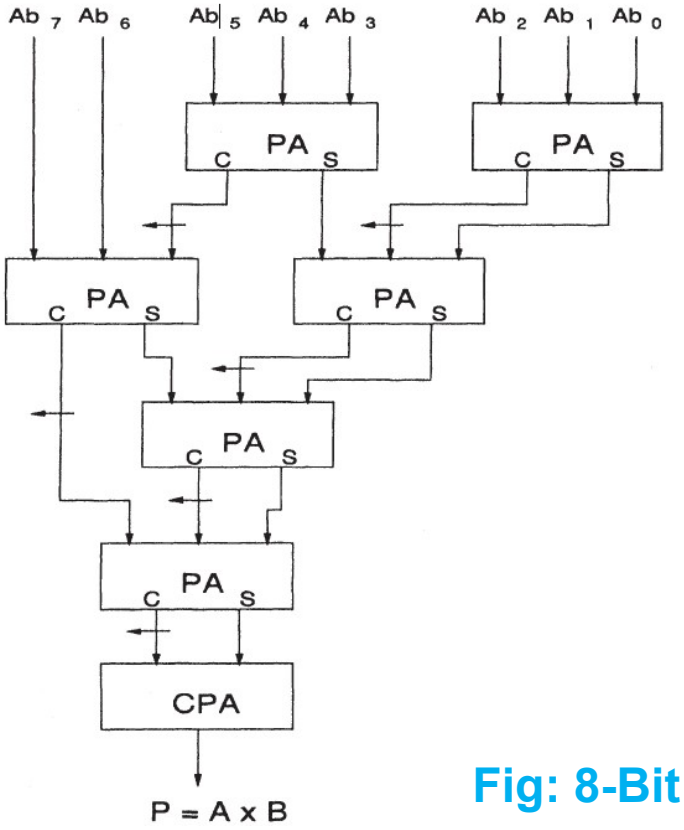
Each cell in the *NXN* array includes an and gate to form the appropriate partial product bit. The cells labeled "G" consist of just the and gate. The cells labeled "FA" are an and gate and a full adder as shown in Figure. The cells labeled “HA" are similar to the full adder cells, but with a half adder. The bottom row of one half adder and *N* - 2 full adders is simply a ripple carry adder. None of the cells in the bottom row include the and gate.

The advantages of the array multiplier are that it has a regular structure and a local interconnect. Each cell is connected only to its neighbors. This translates into a small, dense layout in actual implementations.

The disadvantage is that the worst case delay path of an *NXN* array multiplier goes from the upper left corner diagonally down to the lower right corner and then across the ripple carry adder.

Wallace-Tree Multiplier: is a tree based multiplier which follows hierarchical structure. In this multiplier, partial results are summed in parallel using a tree of adders.

Here pseudo-adders repeatedly used in a tree structure for summing the partial products into 2 larger partial products, and then using a fast carry propagate adder to sum them and produce the product. The structure for such a Wallace tree is shown in Figure



PA➔pseudo-adder

Fig: 8-Bit Wallace Tree Multiplier

Flip-flops and Latches

Flip-flops and latches are used for controlling the timing of computations in synchronous systems.

- **A flip-flop is a circuit which makes the output equal to the input at a clock edge (normally only one of the two clock edges). At any other time, the output is kept constant.**

- **A latch is a circuit which is open during one clock phase (clock low or high) and closed during the other phase. When the latch is open, its output is equal to its input, other wise its output is kept constant.**

CMOS flip-flops can be static or dynamic.

Static flip-flops are of two main types, gates based and transmission gates based.

Dynamic flip-flops are of many types; three main types are, C^2MOS, precharged TSPC (true single phase clock) and non-precharged TSPC.

Latches are normally dynamic: three main types are, C^2MOS, pre charged TSPC, and non-precharged TSPC.

Static flip-flops

Two types of static flip-flops, gate-based and transmission-gate based flip-flops. In gate-based flip-flops figure(a), It is designed from four 3-input gates(CMOS realization) and two inverters, and therefore includes 28 transistors. By assuming a data activity of αa, we may estimate the dynamic power consumption of this flip-flop as follows:

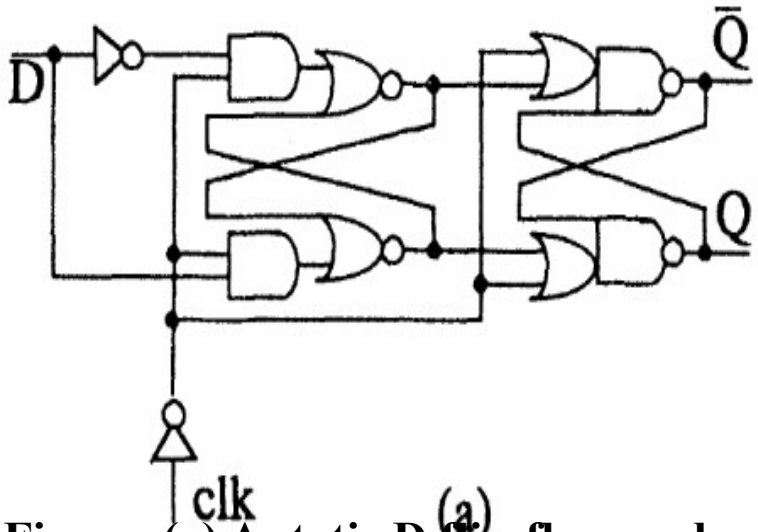


Figure (a) A static D flip-flop made from gates,

We have 4 clocked inputs and 1 inverter, with a total of 10 transistor input capacitances and 2 output capacitances(clocked inv).

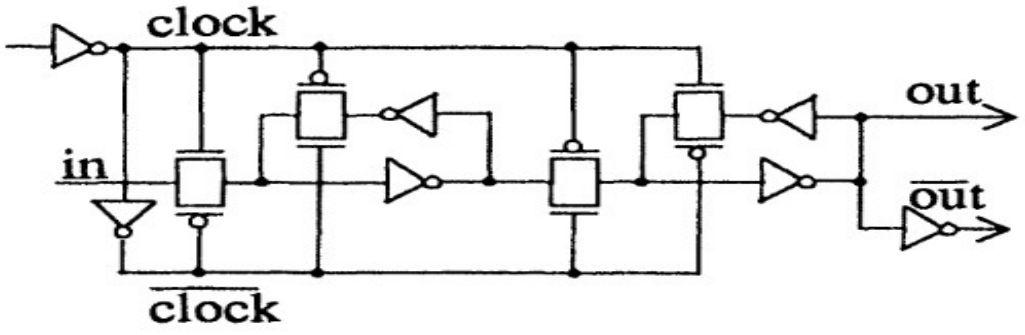
Furthermore, we have two And-Or-Inv, and two Or-And-Inv gates, each gate including 6 transistors, and an inverter.

There are 10 inputs (2 and(4T) +2 or(4T)+1Inv(2T))=10T) (for AND-OR-INV gate) driven by data signals, and also one inverter output (with 2 connected transistors) and 4 gate outputs, each with 6 transistors connected (all the transistors corresponding to output from Vdd to Ground are considered for data output capacitance C0, here for AND-OR-INV output 3 pull up and 3 pull down transistors from Vdd to GND➔6 transistors for each gate structure therefore 6x4=24C0). Thus totally:

$$P_d = 10f_cC_iV_{dd}^2 + 2f_cC_oV_{dd}^2 + (10)(\alpha_a/2)f_cC_iV_{dd}^2 + 2(\alpha_a/2)f_cC_oV_{dd}^2 + 24(\alpha_a/2)f_cC_oV_{dd}^2$$

or

$$P_d = [10C_i + 2C_o + 10(\alpha_a/2)C_i + 26(\alpha_a/2)C_o]\, f_cV_{dd}^2$$



(b)

(b) **A static D flip-flop made from transmission gates.**

In Figure b we have a positive edge-triggered static flip-flop based on transmission gates. This circuit is made from 7 inverters and 4 transmission gates, that is a total of 22 transistors. Following the same procedure as above, we may estimate the power consumption by noting that 12 transistor gates are clocked(2 Inv i/ps and 4 TGs gate i/ps), 2 inverter outputs are clocked, and 5 inverters(5x2=10) input and 5 inverters(5x2=10) output and 4 transmission gates(switch) have a data signal on their inputs and outputs.

This gives a total power consumption of:

$$P_d = [12C_i + 4C_o + 10(\alpha_a/2)C_i + 26(\alpha_a/2)C_o]f_c V_{dd}^2$$

where we have treated transmission gate inputs and outputs as inverter outputs (5 inv+4TG(8Inv))=13x2=26T)from a capacitance point of view (i.e. in TGs data input and outputs are connected to drain and source of transistors, so Ci is considered as C0)

Dynamic Flip-Flops

Dynamic flip-flops are many types. Again, we will assume that only one clock signal is available from outside and that clock buffering is included in each flip-flop. Here 3 types of Dynamic flip-flops are considered.

i) C^2MOS (ii) TSPC (iii) Non-precharged TSPC

i) C^2MOS flip-flops

C^2MOS latch contains 12 transistors, of which 8(2invx2+2P+2n=8) are clocked at their gates and 4 (2 inv o/ps)at their outputs. Furthermore there are 4(2p+2n=4)transistors with data on their inputs and 8(4p+4n) transistors with data on their outputs

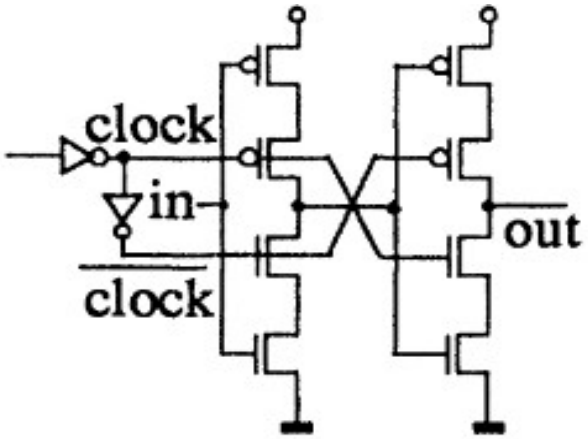


Fig. a

(all the transistors corresponding to output from Vdd to Ground are considered for data output capacitance C0). From this we may estimate the power consumption as:

$$P_d = (8C_i + 4C_o + 4(\alpha_a/2)C_i + 8(\alpha_a/2)C_o)f_c V_{dd}^2$$

ii) Precharged TSPC (True Single Phase Clock) flip-flops

Flip-flop containing 11 transistors, of which 6(2P+2n+1invx2) are clocked on their inputs, 2 are clocked on their outputs, there are 3(p1+n1+n2) transistors with simple data on their inputs, 2(third str top and bottom tr i/p) transistor gates connected to a precharged node, 2 data nodes with 3 transistors(1st and 2nd str) and 1 precharged node with 3 (3rd str) transistors. Totally we estimate the power consumption to:

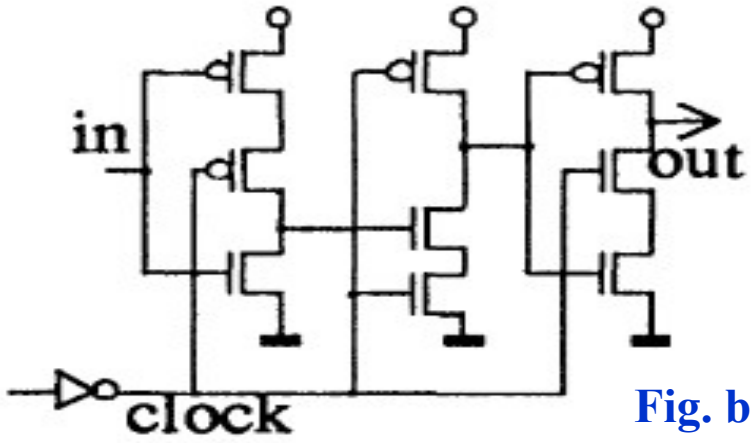


Fig. b

For the output structure(3), the activity factor α0 is not taken in to account because precharged node

$$P_d = (6C_i + 2C_o + 3(\alpha_a/2)C_i + 6(\alpha_a/2)C_o + 2(1/2)C_i + 3(1/2)C_o)f_c V_{dd}^2$$

(iii) Non-precharged TSPC flip-flops

We may also create an edge-triggered flip-flop from two non-precharged TSPC latches. In this case we have 14 transistors, of which 6 are clocked on their inputs, 2 are clocked on their outputs and the others have data signals. From this we get:

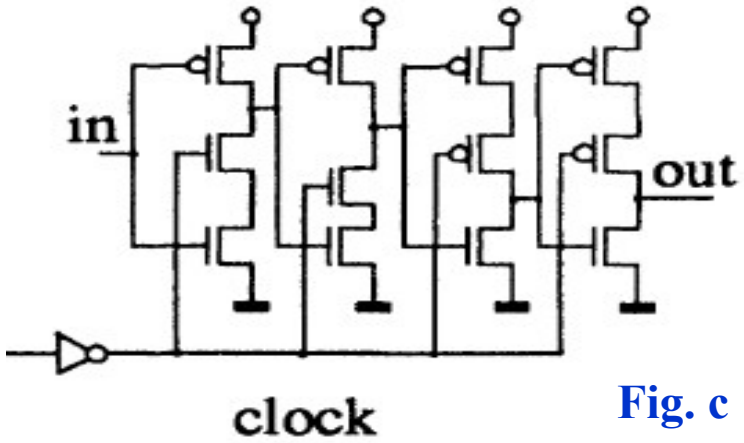


Fig. c

$$P_d = (6C_i + 2C_o + 8(\alpha_a/2)C_i + 12(\alpha_a/2)C_o)f_c V_{dd}^2$$

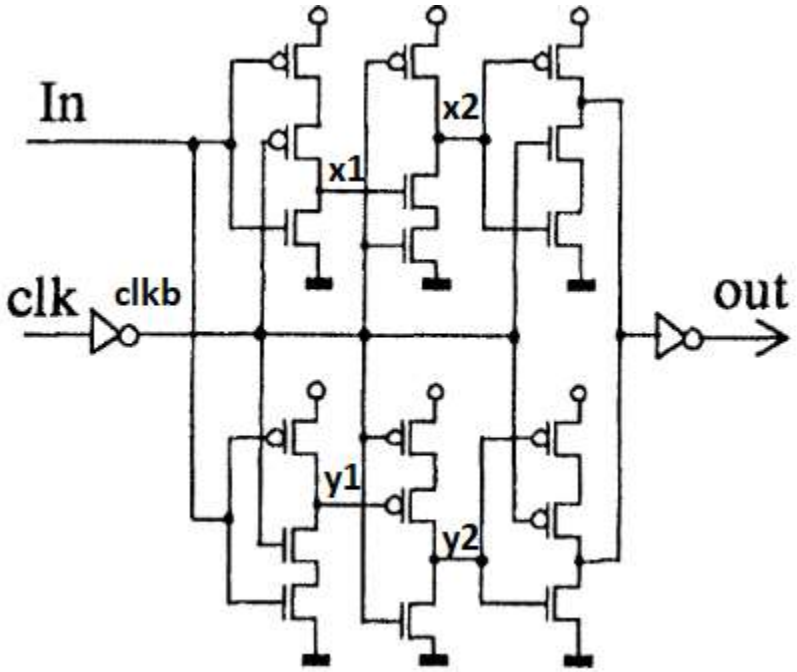
Latches

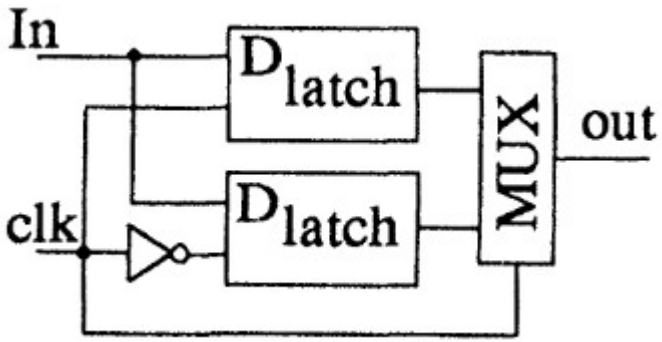
Common latches are normally dynamic. In the dynamic flip-flops discussed above. A C^2MOS latch thus consists of one of the stages from Figure a, a precharged TSPC latch is made from the two last stages in Figure b and a non-precharged TSPC latch from the two first (or the two last) stages of Figure c. The power consumption of these circuits can then be estimated in a similar way.

Double-edge triggered flip-flops

By combining two flip-flops (or latches) and one multiplexer, it is possible to form a double-edge-triggered device (DET), which utilizes both clock phases for timing, Figure a. In Figure b we show an example of a DET flip-flop. Using the same principles as above, we estimate the power consumption of this circuit to:

When Clk=0, Clkb=1, then node y1=Inb and y2=In
node x1=0 or Z, x2=hold mode(previous value)
Out=x2
When Clk=1, Clkb=0, then node x1=Inb and x2=In
node y1=1 or Z, y2=hold mode(previous value)
Out=y2





$$P_d = [10C_i + 2C_o + 4(1/2)C_i + 6(1/2)C_o + 8(\alpha_a/2)C_i + 14(\alpha_a/2)C_o]f_c V_{dd}^2$$

(1Invx2+4p+4n=10) + (1inv(o/p)x2=2) +((2P+2n=4)(data) of 3rd str) + (3p+3n=6) of 3rd str o/p +((2p+2n) 1st str + (1n+1p) of 2nd str+1invx2=8 i/p + 14) o/p

Same way RS FF

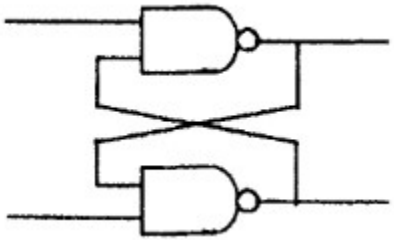
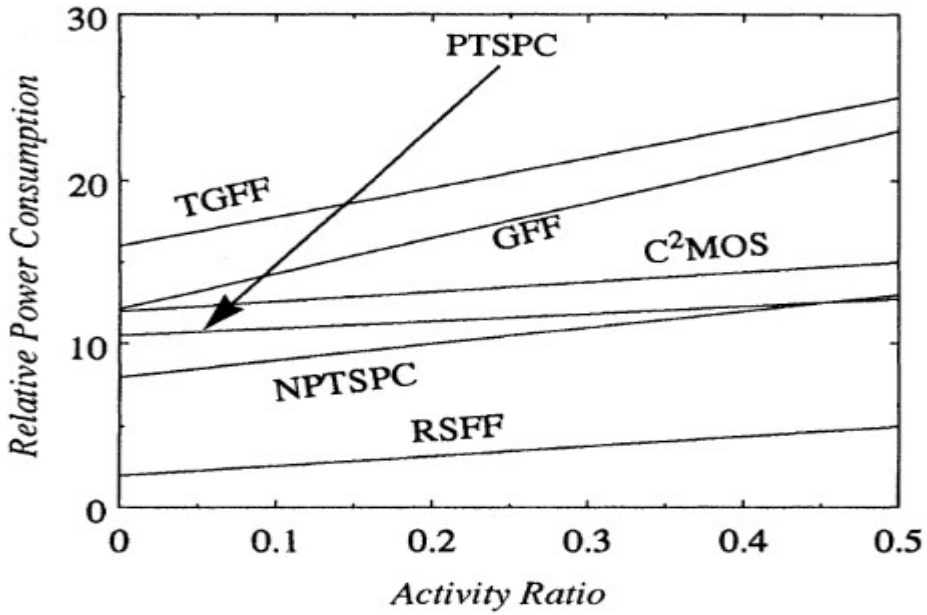


Fig: RS FF

Comparisons between the various techniques



From Figure we can conclude that the gate-based static flip-flop (GFF) has lower power consumption than the transmission gate flip-flop (TGFF), inspite of the fact that it uses considerably more transistors. The reason is that it has fewer clocked transistors.

Of the dynamic flip-flops, the non-precharged TSPC (NPTSPC) consumes least power, The RS-flip-flop latch (RSFF) appears very efficient, but must be considered in connection with the pre charged CVSL logical gate (in this special case the latch is indirectly clocked through the logic).

Considering the double-edge triggered flop-flop (DETFF), the example above has a power consumption which is double compared to the corresponding simple flip-flop (PTSPC in this case). However, for a fixed data throughput it needs only half the clock frequency,

Low Power Digital Cell Library

Over the years, the major VLSI design focus has shifted from masks, to transistors, to gates and to register transfer level. Such upward propagation of design abstraction allows a designer to manage increasingly larger designs. It is becoming time consuming and costlier to design digital circuits manually at the gate-level each time because of the complexity of the chip. Most digital circuits are now synthesized from some form of higher-level description automatically.

The basic building blocks of gate-level synthesis are the gates or cells. The quality of the gate-level circuit synthesis depends on the quality of the cell library. In this section, we will discuss some properties of a cell library that forms the basis for good low power design.

Cell Sizes and Spacing

In the top-down cell-based design methodology, the trade-off among power, area and delay is performed by selecting the appropriate sizes of the cells. Sizing at the transistor level is seldom applied because of the difficulties in automated physical design. Therefore, the most important attribute that constitutes a good low power cell library is the availability of wide ranges of cell sizes for commonly used gates.

For example, if the timing constraint commands the use of a 3X gate and the closest available gate is at 4X, the resultant choice would be IX larger than necessary, thus wasting some power.

The *spacing of the cell sizes should also be carefully* chosen. The *capacitance distribution profile of the circuit should be considered in* drive strength spacing. For example, if most net capacitance is within [0.lpF, 0.5pF], there should be more cell sizes available to drive capacitances at this range. The lower drive cells should be spaced closer than the higher drive cells.

Today, the speed of logic synthesis is still a concern and having too many cells slows down the logic synthesis process. One way to reduce library cell count without too much compromise in quality is to have more size selections for gates that are commonly used.

Varieties of Boolean Functions

WKT for n-input variables there are 2^n entries at the output column of the truth table, which can be assigned with zeros or ones. Each zero-one assignment of the output column results in a unique Boolean function. Therefore, there are 2^{2^n} distinct Boolean functions with n-inputs. The number *M is enormous even for very* small values of *n. For $n \geq 3$, only a small number of M Boolean functions are* available as gates in a cell library. Typically, the Boolean operations are implemented, such as AND, OR, EXOR, AOI, OAI, etc.

The lack of varieties of Boolean functions in a cell library can result in inferior circuits to be generated. For example, if the Boolean function *Y= AB' were to be* implemented and the *inverted-input cells are not available, the logic synthesis system* has to use an INVERTER and an AND gate to implement the function. This is shown in Figure. The two-cell implementation is less power efficient compared to the single-cell implementation because of the external routing capacitance at the output of the inverter.

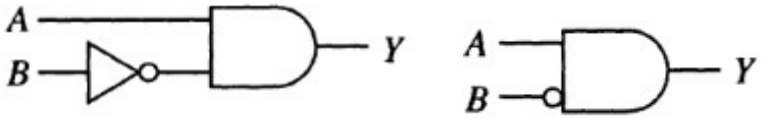


FIGURE: Inverted input cells for low power cell library.

Implementation of all *M Boolean functions in a cell library for more inputs is obviously impossible, but can be implemented as much as possible. T*he functions *Y = AB' and Y = A'B are identical under symmetrical equivalence and there is no* need to implement both.

Two Boolean functions are equivalent under permutation (called *P-equivalent*) *if one function can be obtained from the other by permuting the input* variables. For example, the function *Y= AB + C is P-equivalent to Y = A C + B* because we can interchange the inputs *B and C of one function to obtain the other.* To reduce the cell count, the cell library designer only needs to implement one function from each permutation -equivalence class.

It **is also interesting to examine the equivalent function under *input negation* (called** *input negation equivalent*). **The *function Y = AB is input negation equivalent to Y = A'B and Y = A'B' because we can invert the input variables to obtain the other* functions. Similarly we can also define *output negation equivalent*, *e.g.*, *Y = AB is* output negation equivalent to *Y = (AB)' because one is obtained from the other by* negating the output.

The equivalence relation formed by either input negation or permutation is called *NPequivalent*. *Two functions are NP-equivalent* if one can be obtained from the other by any combination of input negations or variable permutations. Similarly, we define *NPN-equivalence.*

Table 1 shows the number of classes of n-input Boolean function under the various equivalence relation defined above.

TABLE : Classes of Boolean function under P, NP, and NPN-equivalence.

Input var. (n)	Num. func. 2^{2^n}	Strict n variables	P-equiv	NP-equiv	NPN-equiv
1	4	2	2	1	1
2	16	10	8	3	2
3	256	218	68	16	10
4	65536	64594	3904	380	208
5	$\approx 4.3 \times 10^9$	$\approx 4.3 \times 10^9$	$\approx 3.7 \times 10^7$	1,227,756	615,904

The third column gives the number of *n* variable functions that are non-degenerate, i.e., all *n variables are required to deter*mine its output value. The fourth, fifth and sixth columns give the number of distinct functions under P-equivalence, NP-equivalence and NPN-equivalence, respectively, among the non-degenerate functions.

Even with small *n, the number of distinct cells to be implemented under the various* equivalence classes is large. A cell library that covers more classes should produce circuits with better quality because it offers more choices of distinct Boolean functions.

Figure below shows ten types of three-input functions under the NPN-equivalence relation.

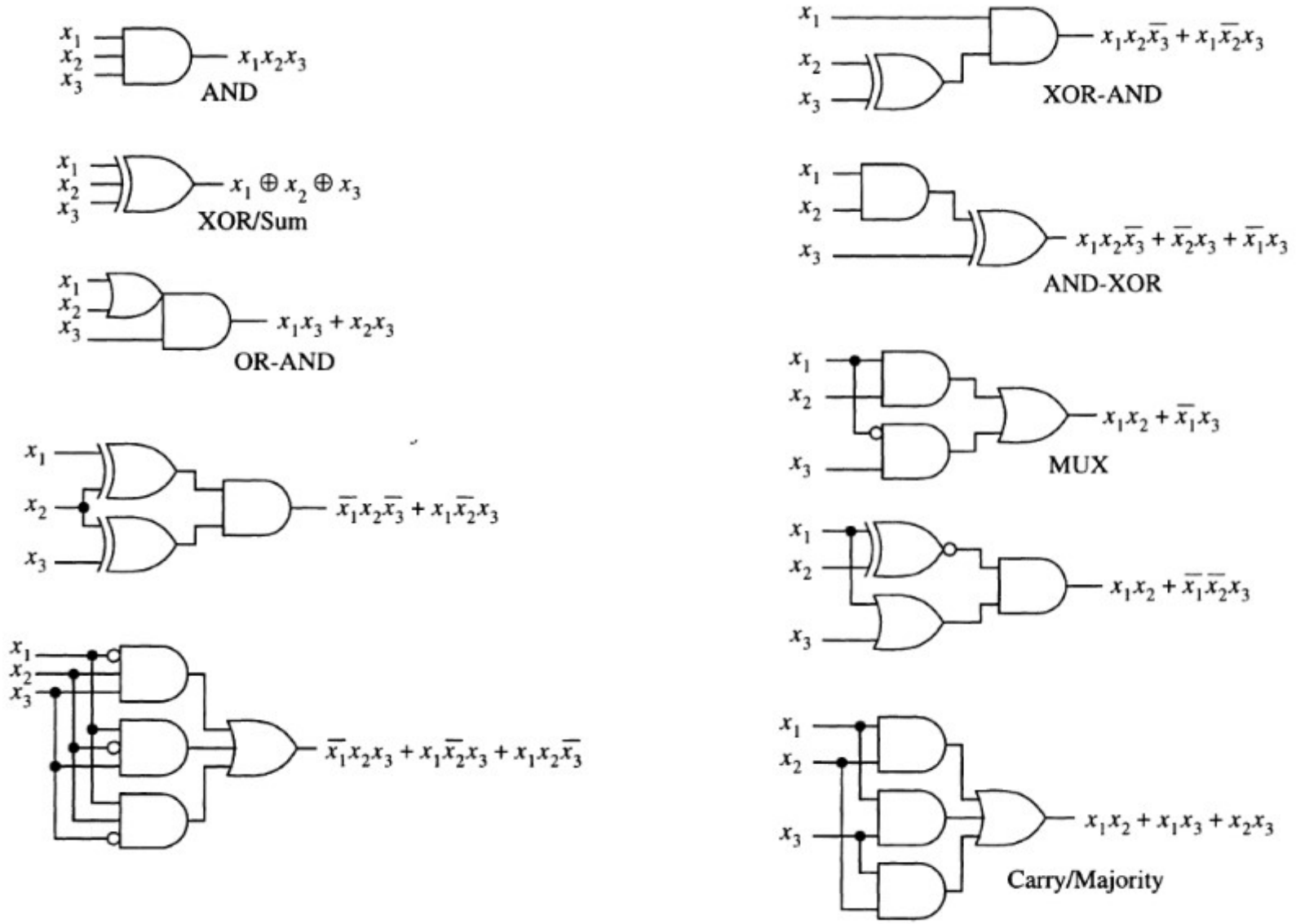


FIGURE: Ten 3-input functions under the NPN-equivalence relation.

Low Power SRAM Architectures

SRAM organization:

The generic RAM organization that illustrates the basic features of most RAMs is shown in Figure1. Which includes the following blocks in a SRAM:

Memory Core, Word Decoders, Column Decoders, Precharge, and Sense Amplifiers

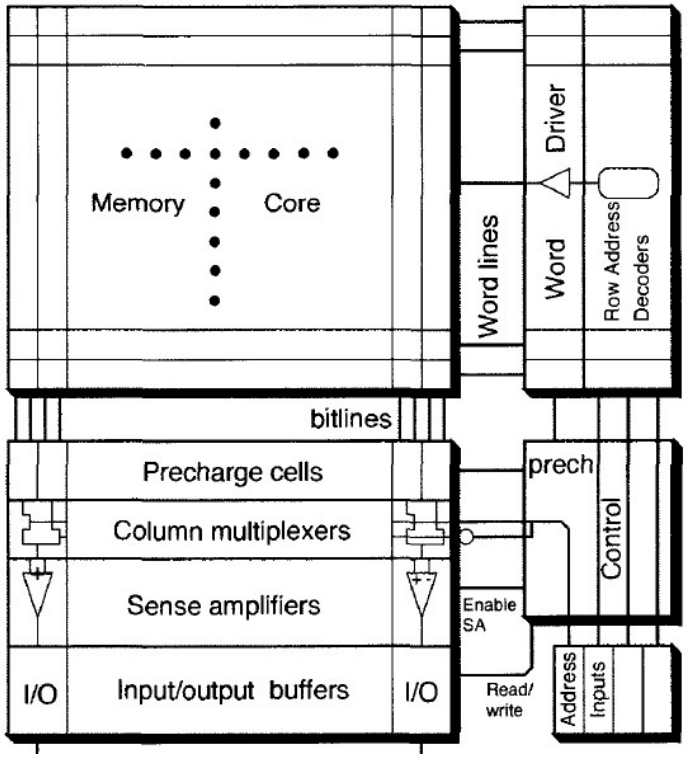


Figure1: Organization of a RAM

***Memory Core*: The actual storage of information is done in a two-dimensional array of memory cells called the memory core. This structure can access any cell by accessing a particular row and a column.**

A row is activated by a global line spanning all cells of the row and is termed the word line. This line enables the cells to be written into or read out. Every column can now be accessed individually through vertical bit lines that span individual columns of the memory array.

Individual cells in the memory core can be accessed by activating a word line and reading data out on a particular set of bit lines.

Word Decoders: The set of cells that generate the word line signals form the word decoders. This structure takes a set of n address lines and generates 2^n word lines. At most one of the word lines is active at a time.

Column Decoders: Column decoders select particular bit lines for being connected to sense amplifiers. This is accomplished either by sensing every bit line and gating a few of them out or by using pass gates to enable them to a few sense amplifier inputs.

Precharge: **Differential read/write schemes are used for memory cells. This requires bit lines to be set up in a well-defined state before an access. Precharge cells accomplish this. Dynamic schemes use a clocked precharge cell to charge bit lines. Static schemes leave the precharge continuously on, in which case they resemble a load cell on the bit lines. Precharge cells also accomplish equalization of differential bit line voltages.**

Sense Amplifiers: **These circuits accomplish the conversion of bit line differentials to logic levels. Sense amplifiers are analog circuits, often resembling simple differential amplifiers. Some of the above components are discussed in detail in the following sections.**

MOS STATIC RAM MEMORY CELL: Two forms of SRAM cells have become popular and are classified based upon the number of transistors in the cell. These are the four-transistor (4T) SRAM cell, which is all NMOS, and the six-transistor (6T) CMOS SRAM cell.

The 4T SRAM Cell

The circuit topology of the 4T NMOS cell is shown in Figure2. Inverters in this cell use an NMOS transistor with a resistive load. The advantage this cell is the high degree of compactness in its layout.

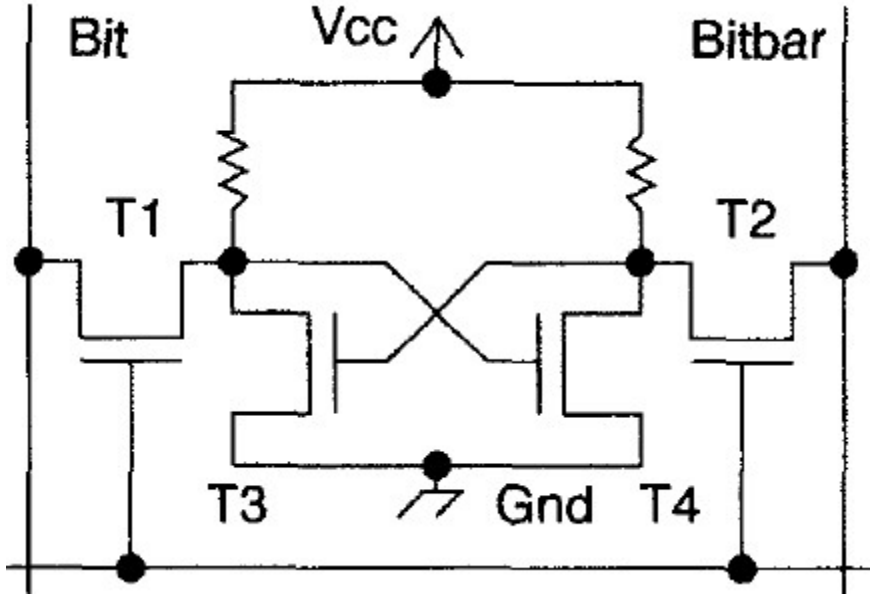


Fig2: Four-transistor SRAM bit cell

A lower transistor count, the absence of n-well areas, and process techniques that allow contacts from active areas of polysilicon all contribute to a lower area.

The cell consumes higher power because of continuous current drawn by the NMOS-only inverters. While it is clear that a higher resistance reduces current consumption and leads to lower standby power consumption, it seriously impacts the noise sensitivity of the cell.

The extreme case of the resistances being absent leads to a 4-T DRAM cell.

The 6T SRAM Cell

SRAM can be in one of three possible states. It can be in the stable state with the cell holding a value or it can be in the process of carrying out a read or a write. The stable state of the cell occurs with the word line connected to low. The cell is effectively disconnected in this mode of operation.

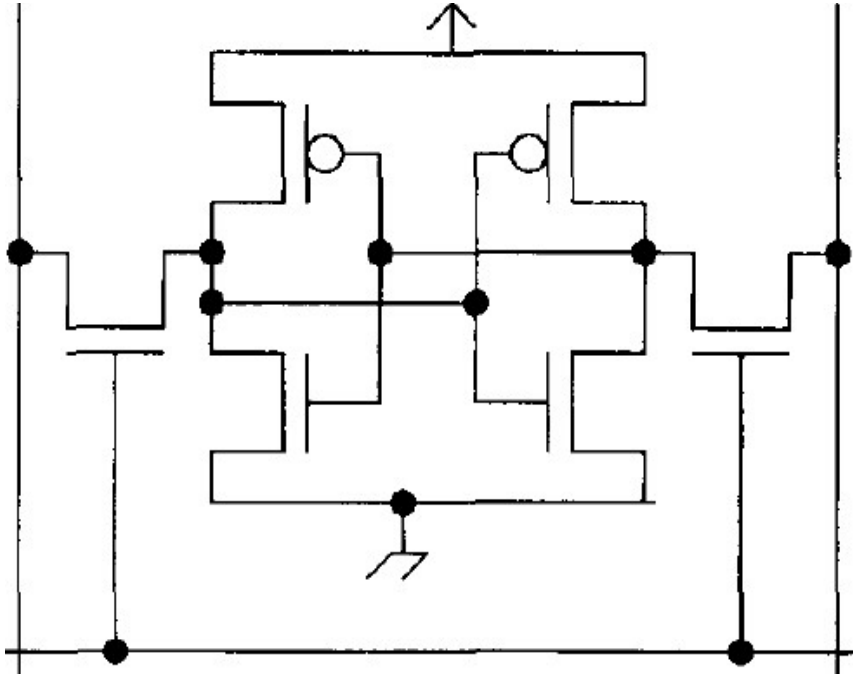


Fig3: Six-transistor RAM cell

A read operation is initiated by precharging the bit lines high and activating the word line. One of the bit lines discharges through the bit cell, and a differential voltage is set up across the bit line. This voltage is sensed and amplified to logic levels. A write to the cell is carried out by driving the bit lines to the required state and activating the word line.

The energy consumption is reduced by reducing one of four factors: total switched capacitance, voltage swing, activity factor, or frequency of operation. In memory cells, it is usually the voltage swing at various points that can easily be manipulated by a circuit designer without impacting other specifications of the memory. It is clear that to achieve the goal for lower power, reduced voltage swings on bit lines and word lines are essential.

BANKED ORGANIZATION OF SRAMs

Banking is an organization technique that targets total switched capacitance to achieve reduced power and improved speed. A memory needs a *R X C* memory core, where *R* is the number of rows and *C* is the number of columns. In general, if C_{cell} is the capacitance of the bit line per cell, a total capacitance of $RxCxC_{cell}$ is switched.

Clearly splitting the memory into a set of smaller memories is an efficient way to reduce the total switched capacitance.

Figure shows the organization of a banked SRAM. An additional set of decoders is now necessary to select one of *B* banks (we have $B = 4$ in Figure). While an additional delay is incurred in selecting the bank, it is offset by the much lower capacitance that is switched per bank. Now the Capacitance is $(RxCxC_{cell})/B$. In actual cases it is possible to find an optimum number of banks given the SRAM size and the form of decoding employed. Banking is clearly a technique to be preferred for large RAMs, as it reduces the overall power consumption by the bank size. Banking also allows some flexibility in SRAM design since failure of a bank still allows one to package a part with a smaller amount of total memory.

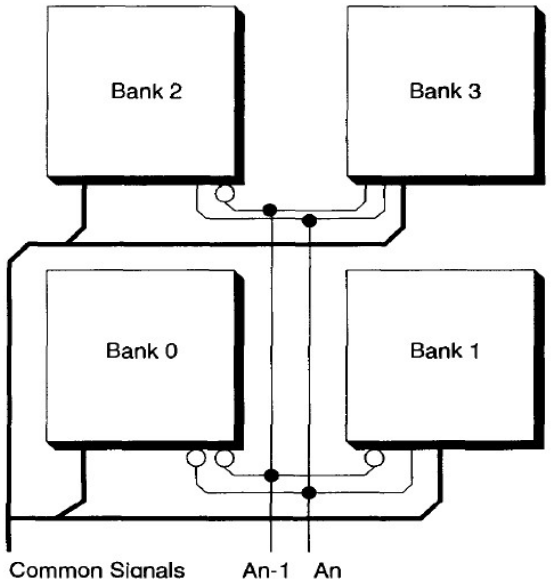


Fig: Banked organization of SRAMs

REDUCING VOLTAGE SWINGS ON BIT LINES

For achieving performance, SRAMs use a set of sense amplifiers to detect differential voltages developed across bit lines. A read operation in an SRAM can be reduced as soon as this detection is accomplished. This technique of limiting the voltage swing on the bit lines allows saving a fraction of the power expended in the core during reads.

In general, if ΔV is the voltage swing across bit lines, V_{core} is the supply voltage to the core, *r* is the fraction of operations that are reads, and f is the frequency of core operations, then we can model the power expended during reads as $(C_{eff} V_{core} \Delta V r f)/2$.

Figure illustrates this technique and shows waveforms for a RAM with limited bit cell swings.

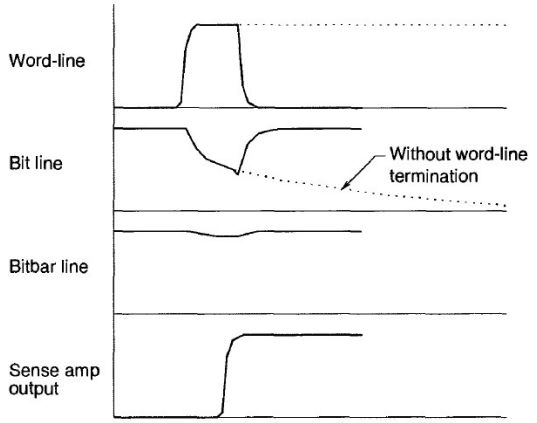


Fig: Early termination of word lines to reduce bit line swing

To limit the bit line voltage discharge and to enable word lines for precise bit cell voltage discharge. A circuit that accomplishes this using a pulse generator is shown in Figure. This circuit gates the word line and the sense amplifiers by a pulse generated using delay cells.

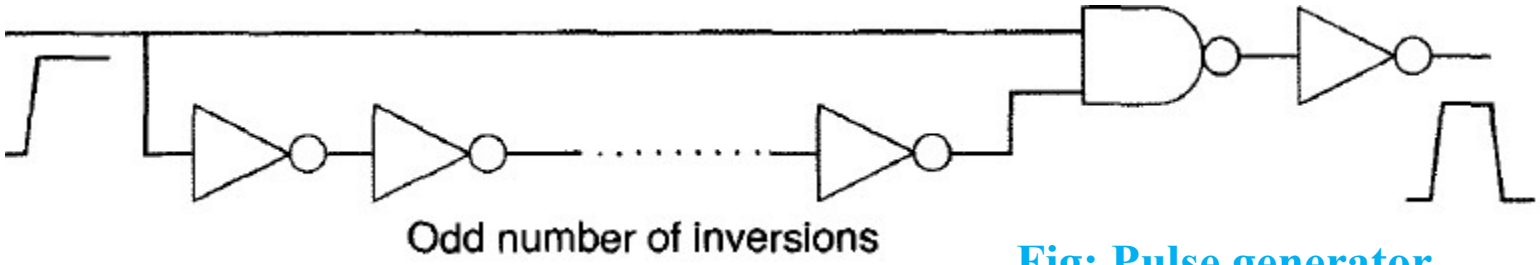


Fig: Pulse generator

In general, the speed of access to various rows is not identical. Clearly the rows closest to the sense amplifiers should give the fastest access times. Similarly columns closest to the word line drivers are enabled first. To utilize a pulsed word line to its best advantage, we can adjust the width of the pulse according to the access time of the RAM.

Precharge Voltage for Bit Lines

SRAMs employ one of two forms of precharge techniques. The first technique uses a static MOS device functioning as a resistive load cell. Enhancement mode NMOS devices operating in their active region, depletion mode NMOS, or standard PMOS devices operating in the saturation region are employed.

Reducing the precharge voltage clearly reduces the power consumption, since the effective voltage swings on bit lines is reduced. Precharge through enhancement mode NMOS devices is more effective from a power viewpoint.

A reduced aspect ratio may be used due to the reduced precharge voltage. This results in cell size reduction. It is possible to get over the above problem of reads corrupting the cell state by employing different voltages on the word line for read and write. In particular, we arrange circuitry to ensure that a lower voltage on the word line is employed for the read as opposed to that for the write.

REDUCING POWER IN THE WRITE DRIVER CIRCUITS

Here power reduction of write driver circuits, namely, the word line drivers and word line decoders are considered. Word line drivers are usually simple buffers designed to fit in the row pitch of a memory cell. Little opportunity for power optimization is present in these circuits.

In general, the row decoding to be as fast as possible, since it directly affects access time for a RAM. There are variety of row decoder structures, the NOR-type decoder and the NAND-type row decoders are considered.

A NAND decoder changes the output of the decoder along one row. Its structure is shown in Figure.

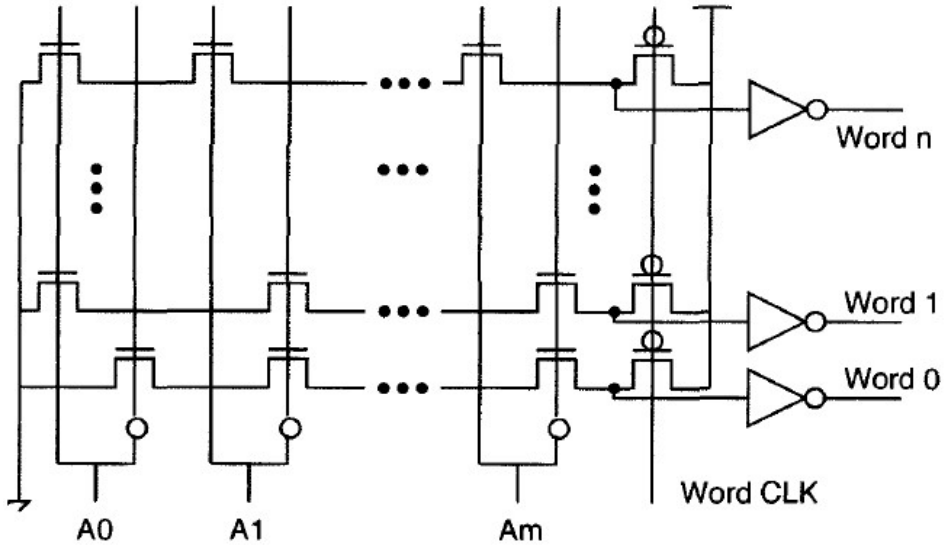


Fig: Domino NAND decoders

In contrast, a NOR decoder activates the outputs of all but one row. Its structure is shown in Figure. A NOR decoder is consequently faster because of the smaller height of the evaluation NMOS stack. The activity of signals is much higher in a NOR decoder, resulting in a higher power consumption than in NAND decoders. NAND form decoder is desirable from a power perspective.

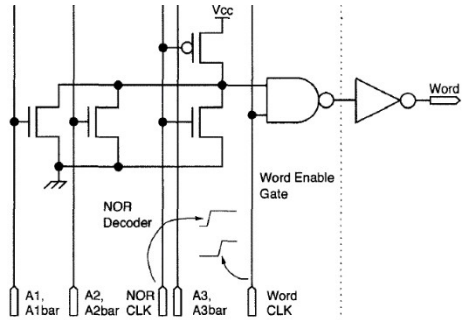


Fig: NOR decoder

Instead of decoding A address lines into one of 2^A word lines, we can split the decode process into a number of steps. Specifically, we first decode $A_1<A$ address lines and use the 2^{A1} lines to activate one of a set of second- stage decoders. The second-stage decoders now decode $A - A_1$ lines into 2^{A-A1} lines. In all we have 2^{A1} second-stage decoders, giving us a total 2^{A1} X $2^{(A-A1)} = 2^A$ lines. This procedure can be repeated to give us a tree structure with a number of intermediate decoders.

REDUCING POWER IN SENSE AMPLIFIER CIRCUITS

The function of the sense amplifier is to amplify small differential bit line voltages into logic levels. This operation should be performed as fast as possible.

Some of the common sense amplifiers used for SRAMs are either simple differential amplifiers or charge amplifiers that are similar to bit cells. A natural trade-off occurs between speed and power for such sense amplifiers. Larger currents improve sense amplifier speed. Sense amplifiers are enabled by a sense amplifier enable signal.

Schemes employed for reducing the power requirements of sense amplifiers based upon the point at which they are activated. The first form limits sense amplifier currents by precisely timing the activation of the sense amplifier for just the period required. The second scheme employs sense amplifiers that automatically cut off after the sense operation.

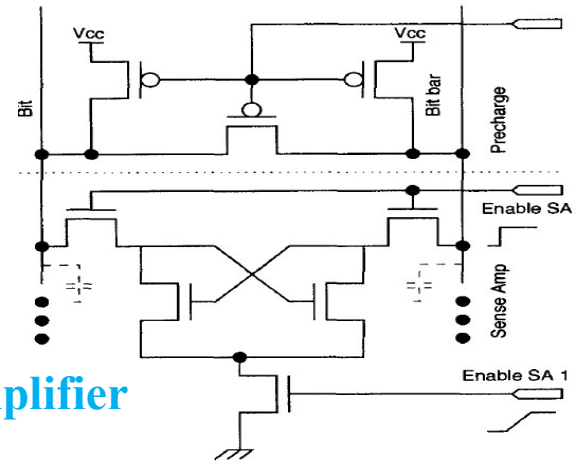


Fig: Differential Charge amplifier

The self-timed RAM core can be extended for obtaining the sense amplifier enable signal. The enable sense amplifier signal is used to set up an SR flip-flop in the set state. Once the dummy sense amplifier has finished sensing, it resets the SR flip-flop, which in turn disables the enable for the sense amplifiers. We rely on every sense amplifier having completed the sense action before the dummy sense amplifier.

Self-latching sense amplifiers accomplish an automatic limiting of currents after sense.