ELH 4.1 VLSI DESIGN

UNIT-III

- Subsystem design and layout
- Reliability and Testing of VLSI Circuits

Subsystem design and layout

Subsystem Design

Introduction

- *A digital system contains several functional blocks, which are known as subsystems.*

- For example, an arithmetic logic unit (ALU) contains functional blocks, or subsystems such as adders, subtractors, multipliers, dividers, comparators, and shift registers. In this chapter, we shall learn how to design such subsystems.

Subsystem Design

Introduction Cont....

- For designing digital systems in MOS technology there are two ways of building the circuits. They are Switch logic(Pass transistor,TG) and Gate logic (NMOS/COMS/BiCMOS) deals with designing of subsystems, which is a small part in a larger system called *leaf cell*.

- The most basic leaf cell of any digital system is the logic gates and these are seen in different technologies like nMOS, CMOS and BiCMOS.

- While designing high regularity should be maintained. This indicates detailed designing of few leaf cells which can be replicated and interconnected to form system.

1. Switch logic

- The switch logic is built on 'pass transistors' or on 'transmission gates'. Switch logic is fast for small arrays and draws no static current from the supply rails. Hence power dissipation of such circuits is small because current flows only on switching.
- Pass transistor can be used as a switch in passing the signals. Switch logic arrangements using basic OR and AND connections along with other arrangements

2. Gate logic.

- Gate logic is based on the general arrangement of inverter as it is the simplest gate. The inverter can be constructed with nMOS, CMOS or BiCMOS technology. Similar to this NAND and NOR can be constructed. Also AND and OR can be constructed with an inverter for NAND and NOR respectively.

The whole design process will be assisted if considerable care is taken with:

- Partitioning of the system so that there are clear subsystems with minimum interdependence and minimum complexity of interconnection between them.

- The design within the subsystems should be simple which helps in cellular design concept. This helps the systems in having few standard cells which are replicated to form high regular structures.

General Considerations of Subsystem Design:

- Lower unit cost.
- Higher reliability.
- Lower power dissipation, lower weight and lower volume.
- Better performance.
- Enhanced repeatability.
- Possibility of reduced design/development periods.

Architectural Issues

- As the complexity of a system increases, the design time also increases. *Thus while designing we have to adopt those design methodologies which allows handling complexity with reasonable time and reasonable amount of labor.*

The following are the guidelines/architectural issues that needs to be considered while designing of the system.

1. Define the requirements carefully and properly.
2. Partition the overall architecture into appropriate subsystems.
3. Communication paths should be carefully selected in order to develop sensible interrelationships between the subsystems.
4. Draw the floor plan of how the system is to map onto the silicon.
5. Aim for regular structures so that design is largely a matter of replication.
6. Draw suitable stick or symbolic diagrams of the leaf-cells of the subsystem.
7. Convert each cell into layout.
8. Carry out design rule check carefully and thoroughly.
9. Simulate the performance of each cell/subsystem.

Some Problems of VLSI System Design

1. How to design complex systems in a reasonable tune & with reasonable effort.

2. The nature of architectures best suited to take full advantage of VLSI and the technology.

3. The testability of large/complex systems once implemented on silicon.

Some Solutions…

Problem 1 & 3 are greatly reduced if two aspects of standard practices are accepted:

1. a) *Top-down design approach* with adequate CAD tools to do the job

   b) Partitioning the system sensibly

   c) Aiming for simple interconnections

   d) High regularity within subsystem

   e) Generate and then *verify each section of the design.*

2. Devote significant portion of total chip area to test and diagnostic facility

3. Select architectures that allow design objectives and high regularity in realization

Structured Design(Top-down) Approach

- Structured design is based on ‘divide and conquer’ strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

- A structured and orderly approach to system design is highly beneficial and becomes essential for large systems.

- Structured design begins with the concept of hierarchy

- It is possible to divide any complex function into less complex sub functions that is up to leaf cells ,*Process is known as top-down design*

Structured Design Approach

- Classical techniques for reducing the complexity of IC design are:

1. Hierarchy
2. Regularity
3. Modularity
4. Locality

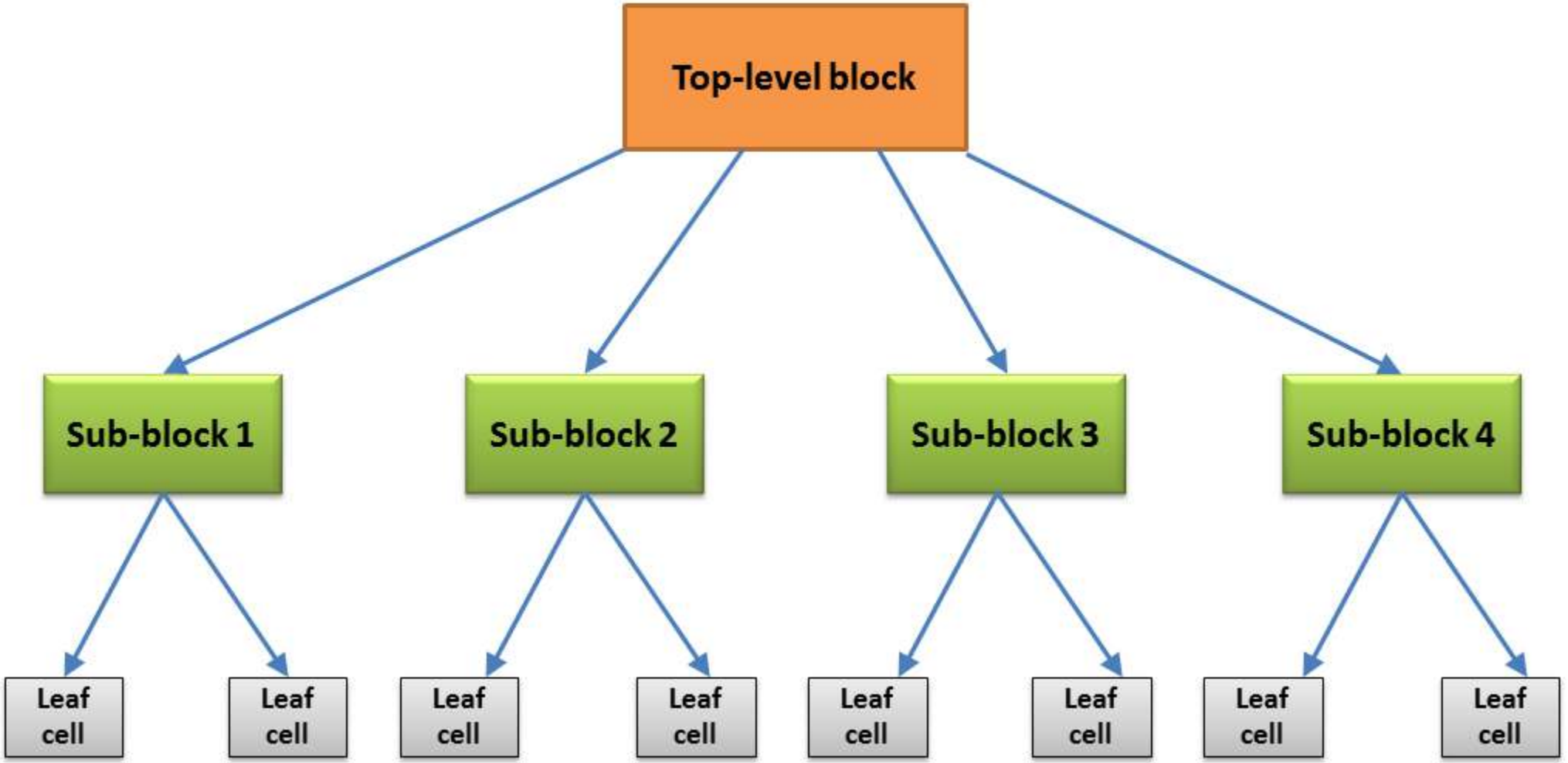
1. Hierarchy: “Divide and conquer “ technique involves dividing a module into sub-modules and then repeating this operation on the sub-modules until the complexity of the smaller parts becomes manageable.

2. Regularity: The hierarchical decomposition of a large system should result in not only simple, but also similar blocks, as much as possible. Regularity usually reduces the number of different modules that need to be designed and verified, al all levels of abstraction.

3. Modularity: The various functional blocks which make up the larger system must have well-defined functions and interfaces

4. Locality: Internal details remain at the local level. The concept of locally also ensures that connections are between neighboring modules, avoiding long-distance connections as much as possible.

Top-down Design Methodology



Top-down Design Methodology

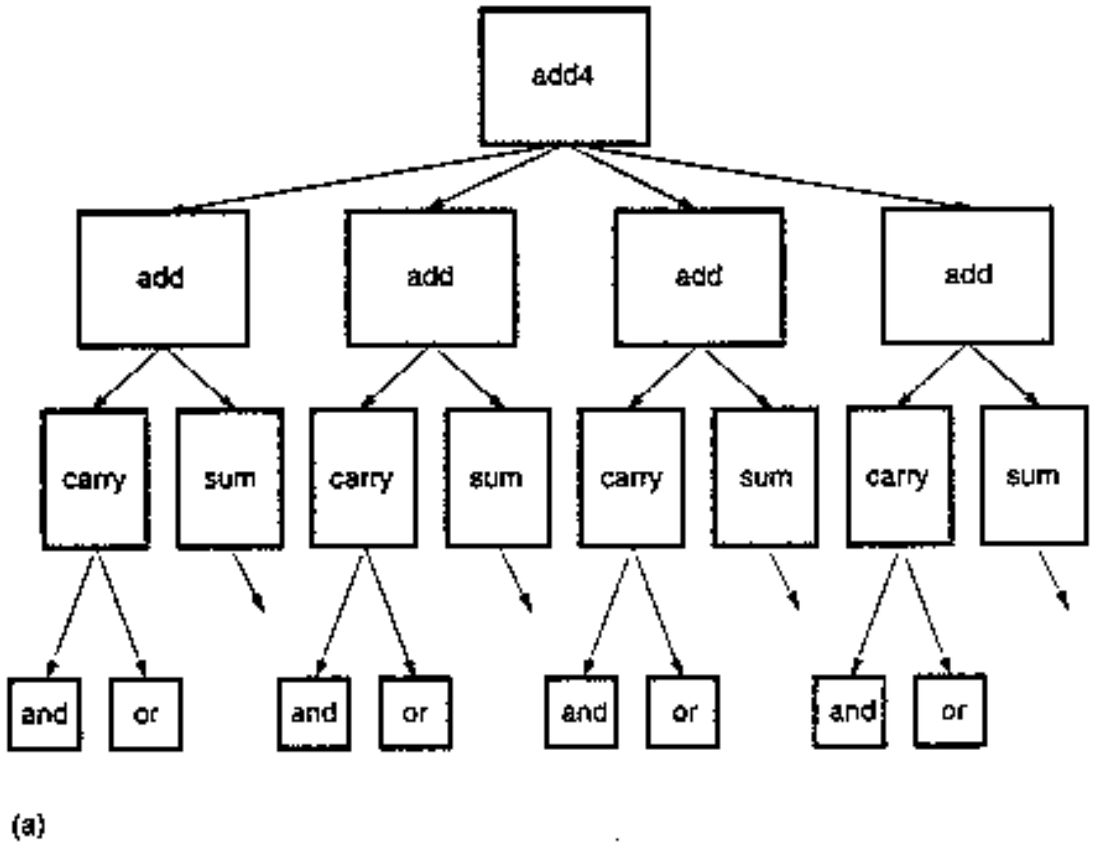
- *In a top-down design methodology, we define the top-level block and identify the sub-blocks necessary to build the top-level block.*

- We further subdivide the sub-blocks until we come to leaf cells, which are the cells that cannot further be divided.

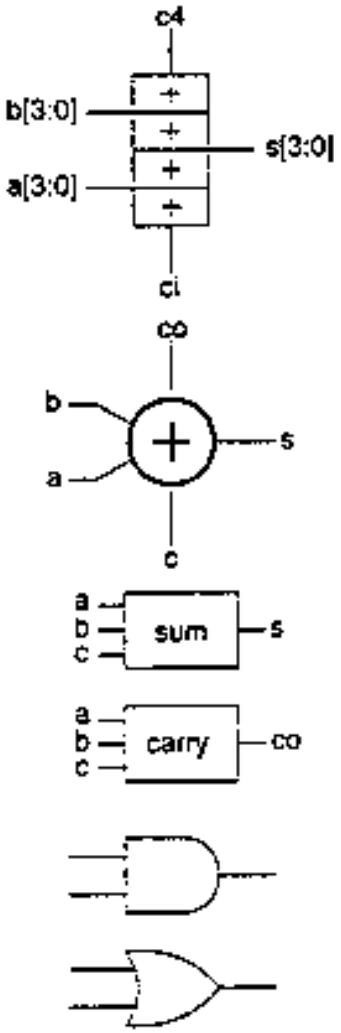
- **For example**; A 4-bit parallel adder is to be designed. The logic designers would break the 4-bit parallel adder to four 1-bit full adders. Each full adder would be broken into two half adders and an OR gate. Each half adder would again be divided into one XOR gate and one AND gate. At the same time the Circuit designer would design the optimized low leakage logic gates: AND, OR and XOR from the transistor / Switch level.

Structural decomposition of a 4-bit adder into its components

Design Hierarchy



(a)



Concepts of Regularity, Modularity and Locality

The hierarchical design approach reduces the design complexity by dividing the large system into several sub-modules. Usually, other design concepts and design approaches are also needed to simplify the process. Regularity means that the hierarchical decomposition of a large system should result in not only simple, but also *similar blocks, as much as possible*. A good example of regularity is the design of array structures consisting of identical cells - such as a parallel multiplication array. Regularity can exist at all levels of abstraction: At the transistor level, uniformly sized transistors simplify the design. At the logic level, identical gate structures can be used,

Examples: Structured Design(Top-down) Approach combinational logic circuits.

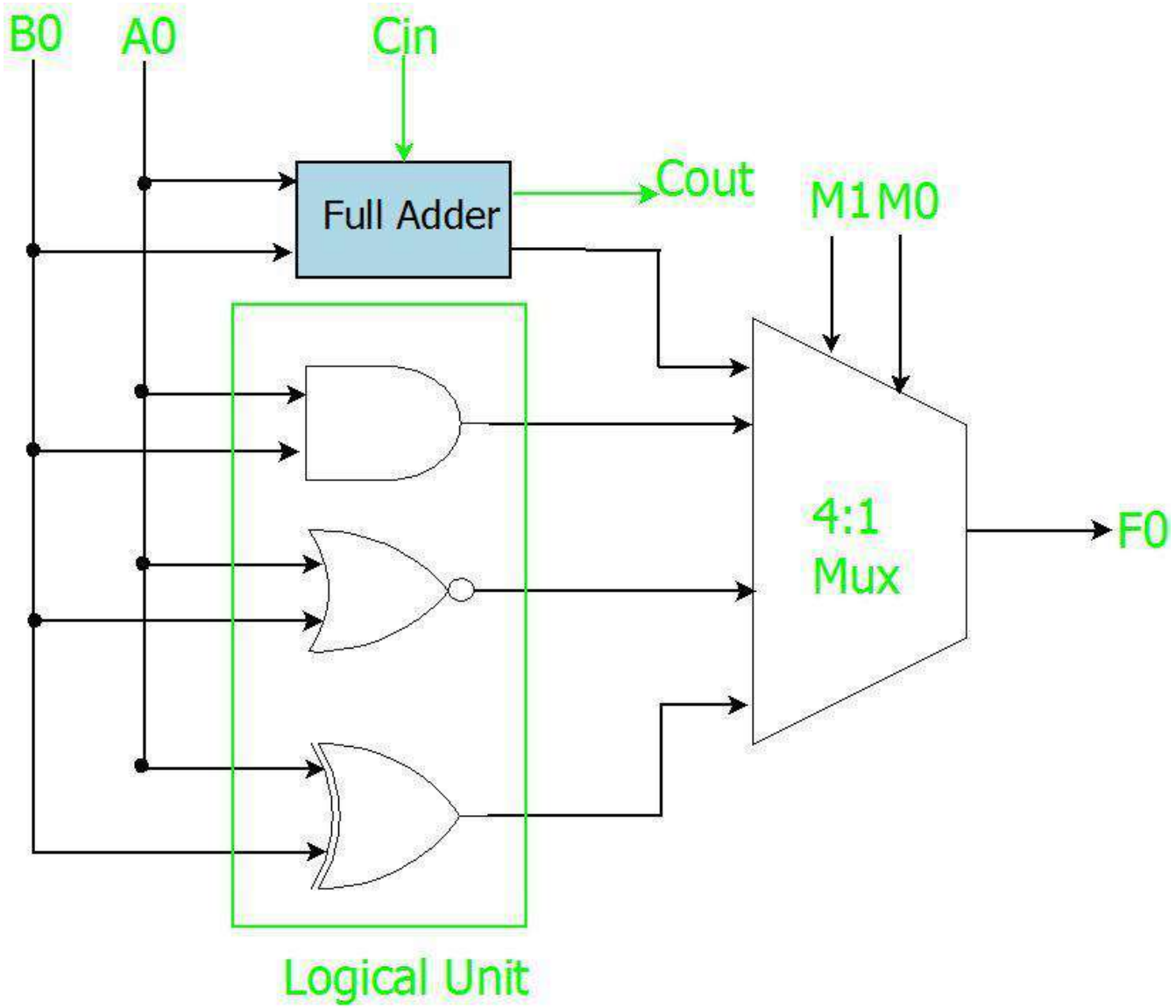
1. 4 bit adder
2. 4 bit ALU to perform the SUM,AND,OR, XOR Operations
3. 2:1, 4:1, 8:1 MUX
4. Parity Generator

Design of 4-bit ALU for AND, OR, XOR, and ADD operations

- An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operations. It represents the fundamental building block of the central processing unit (CPU) of a computer. Modern CPUs contain very powerful and complex ALUs.

- The purpose of the ALU is to perform mathematical operations such as addition, subtraction, multiplication and division. Additionally, the ALU processes basic logical operations like AND/OR calculations. Also known as the arithmetic logic unit, it serves as the computational hub of the Central Processing Unit (CPU) for a computer system

- Lets construct a simple ALU that performs a arithmetic operation (1 bit addition).
- and does 3 logical operations namely AND, NOR and XOR as shown below.
- The multiplexer selects only one operation at a time.



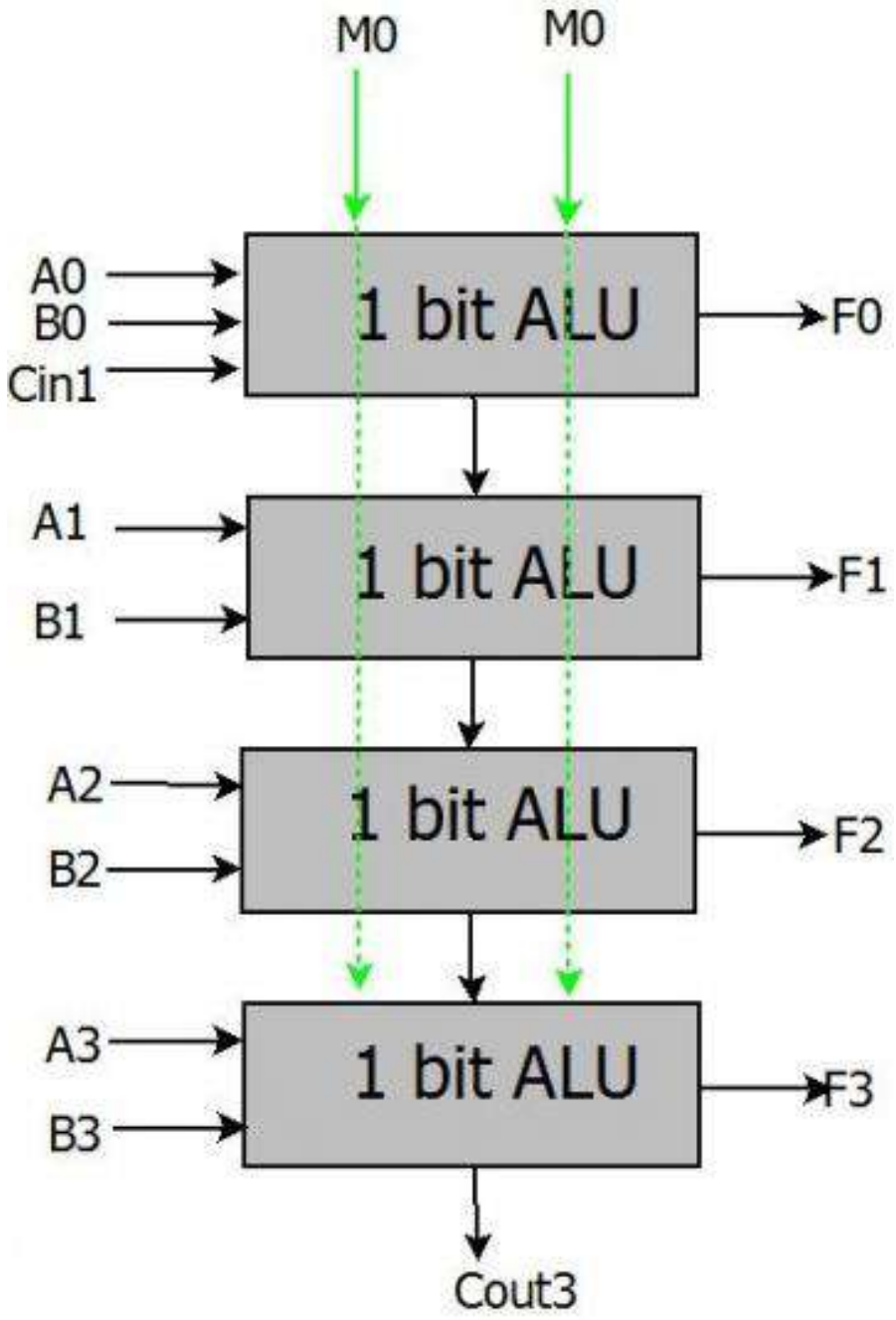
The operation selected depends on the selection lines of the multiplexer as shown in the truth table.

Inputs		Outputs
M1	*M0*	*Operation*
0	0	SUM
1	0	AND
0	1	OR
1	1	XOR

- Now we can take up the 1 bit ALU as block and construct a 4 bit ALU, which performs all the functions of the 1 bit ALU on the 4 bit inputs.

- *Thus a single building block can be constructed and used recursively.*

- The inputs A and B are four bits and the output is 4 bit as well. Figure illustrates it:



Full adder using two Half adders and an OR gate

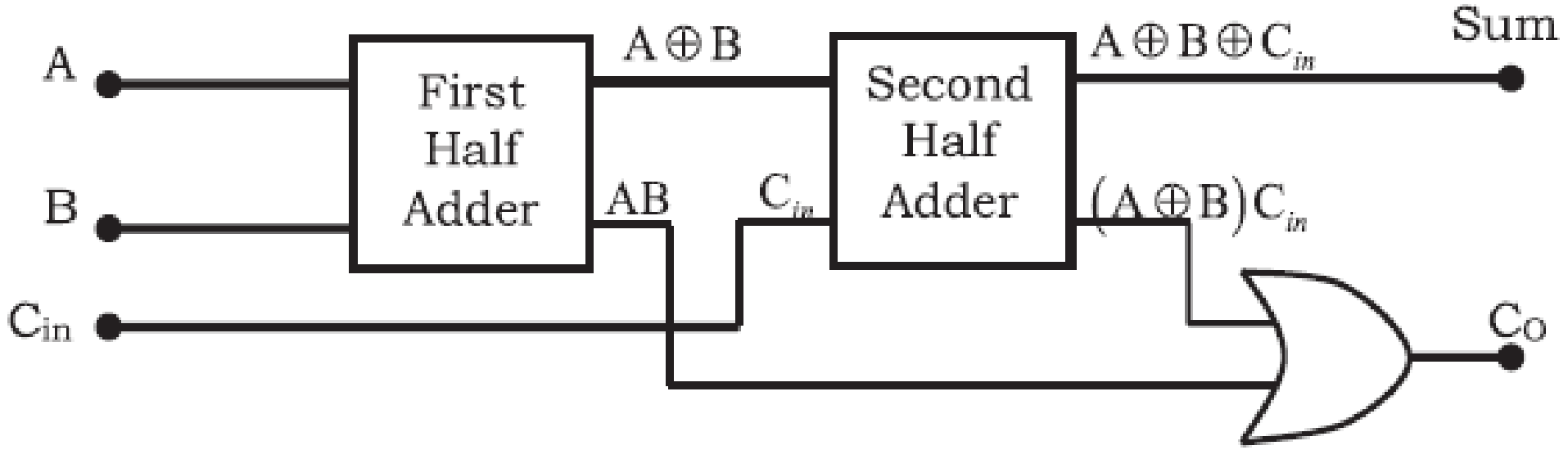


Figure 10.3.8 Implementation of Full adder using two Half adders and an OR gate.

Full adder using two X-OR gates, two AND gates and an OR gate.

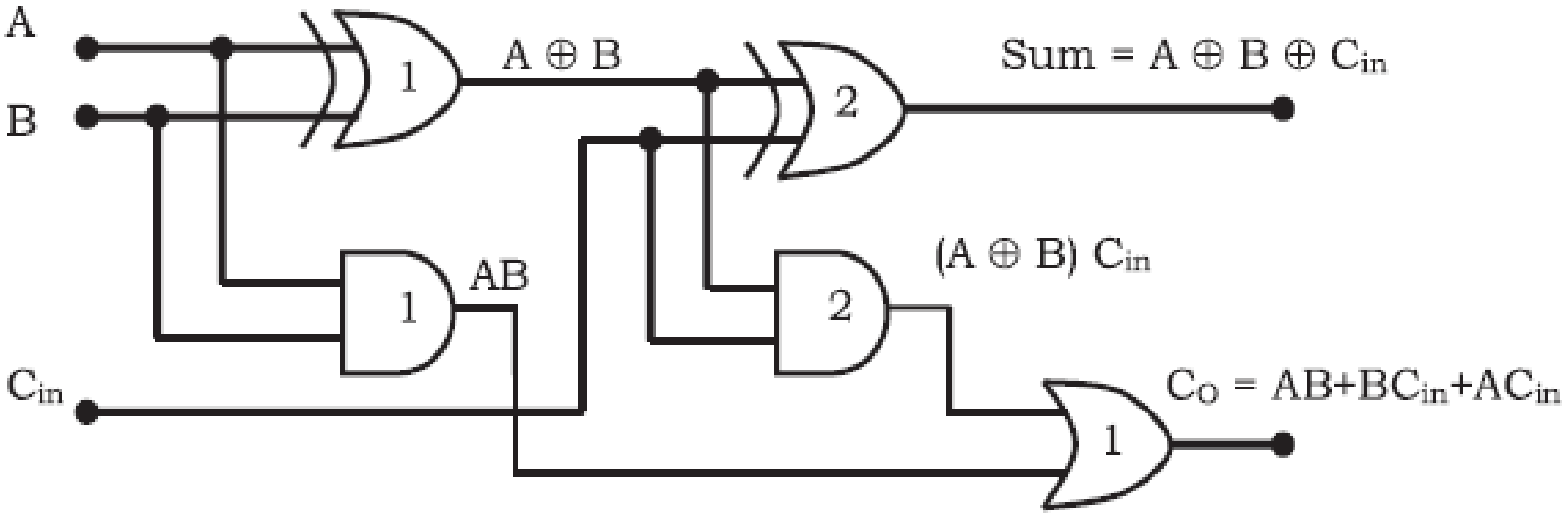


Figure 10.3.9 Implementation of Full adder using two XOR gates and basic gates

There are a few important takeaways here:

•*The selection lines MO and M1 select the function ALU performs*. These selection lines combined with the input arguments and desired functions a Instruction Set can be formed.

•These Instructions can used to create meaningful programs. Since these are required to be easily available they can be stored on ROM unit.

•The input arguments A and B are often stored in Internal Registers. These along with other special purpose register form the registers of the microcontroller.

•ROM memories are slower in speed, hence an intermediate high speed RAM is often used.

•All the critical timings, decoding of the instructions are often grouped together in seperate control and timings unit'

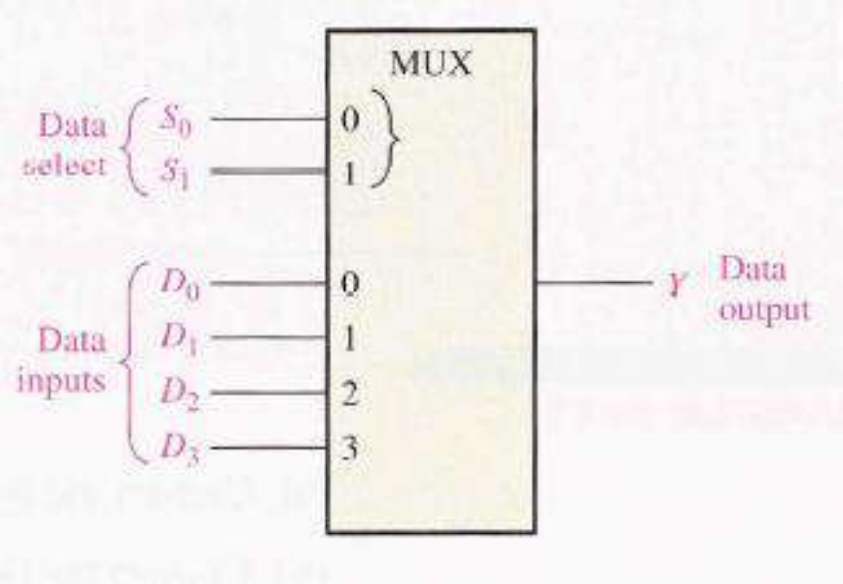
•If a Micro controller would be constructed only from ALU, RAM, ROM there would not be any external interface. Hence we have Input/Output IO ports.

•Additional features such as 'Interrupts, communication protocols, EEPROM, Timers/Counters, Debug interfaces etc are incorporated to make a controller complete.

Multiplexer/data selectors :

- A multiplexer or mux is a combinational circuits that selects several analog or digital input signals and forwards the selected input into a single output line.

- A multiplexer of 2^n inputs has n selected lines, are used to select which input line to send to the output.



DATA-SELECT INPUTS S_1	S_0	INPUT SELECTED
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Now let's look at the logic circuitry required to perform this multiplexing operation. The data output is equal to the state of the *selected* data input. You can therefore, derive a logic expression for the output in terms of the data input and the select inputs.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: $Y = D_0\overline{S}_1\overline{S}_0$.

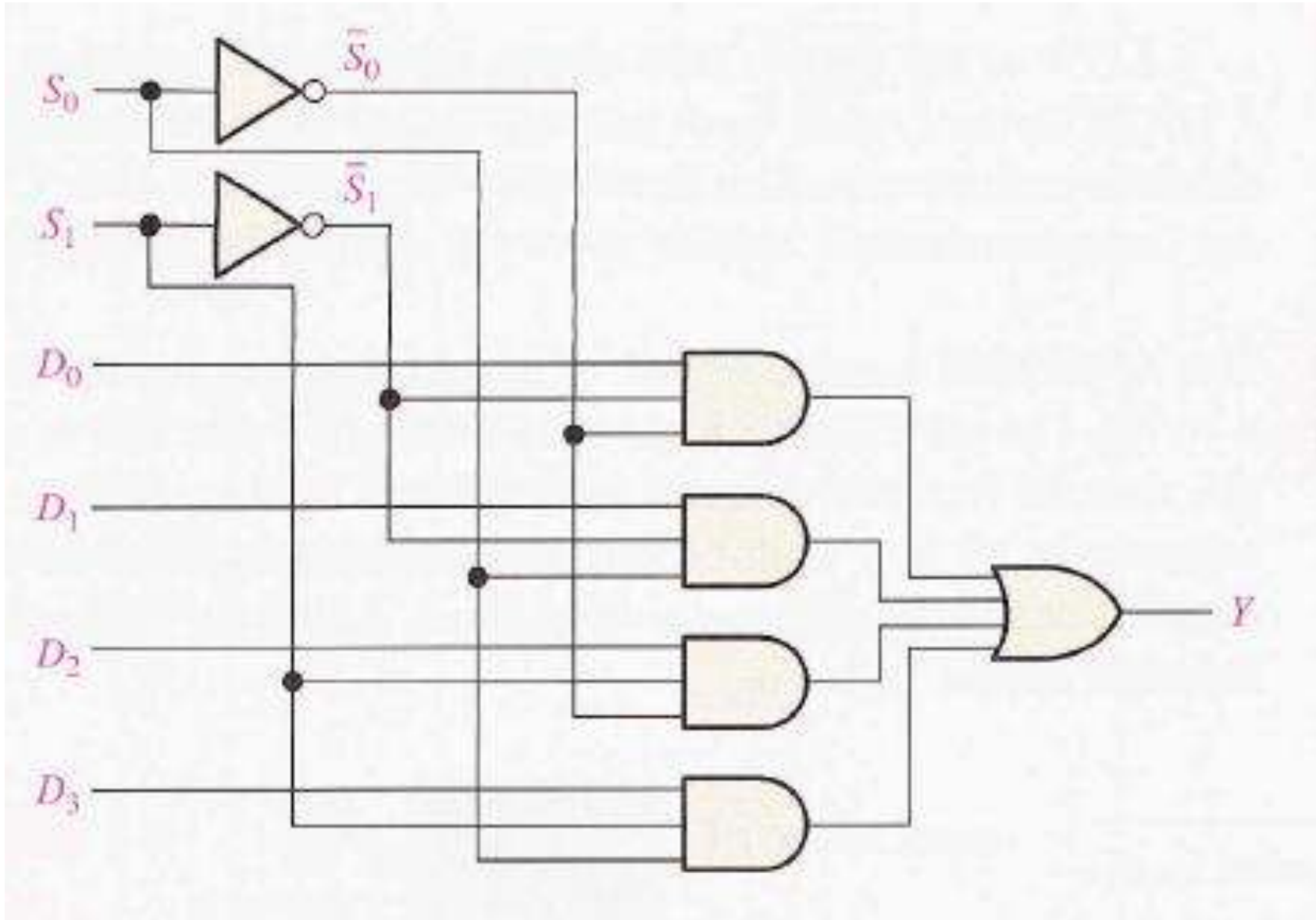
The data output is equal to D_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = D_1\overline{S}_1 S_0$.

The data output is equal to D_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = D_2 S_1\overline{S}_0$.

The data output is equal to D_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = D_3 S_1 S_0$.

When these terms are ORed, the total expression for the data output is

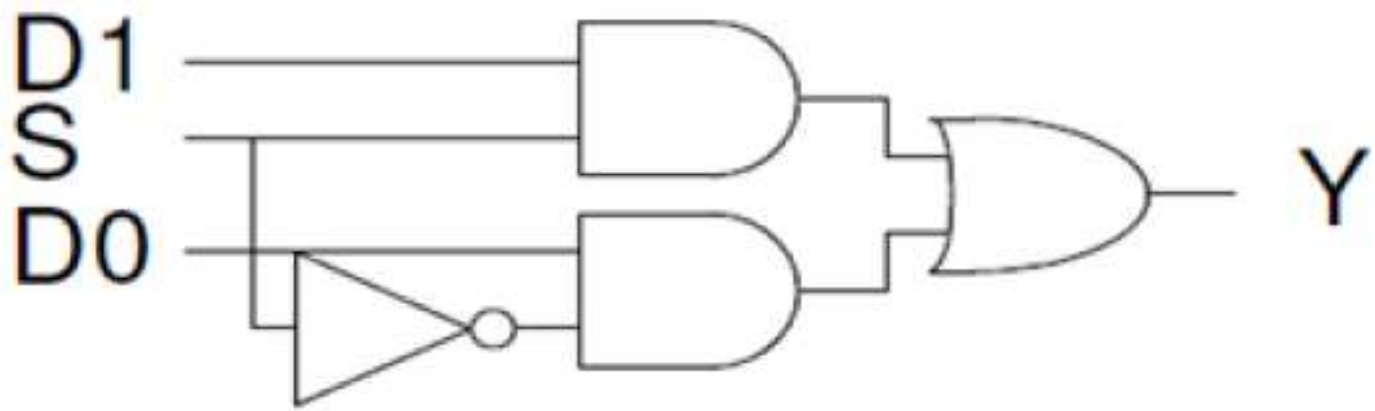
$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1 S_0 + D_2 S_1\overline{S}_0 + D_3 S_1 S_0$$


S_0
$\overline{S}_0$
S_1
$\overline{S}_1$
D_0
D_1
D_2
D_3
Y

Structured Design-Mux Design.. Gate-Level

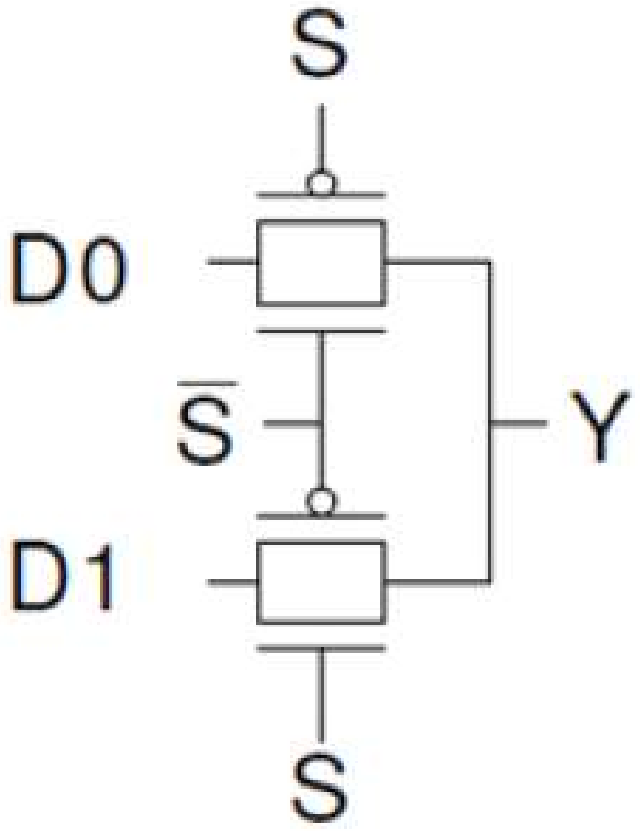
$$Y = SD_1 + \overline{S}D_0 \text{ (too many transistors)}$$

- **How many transistors are needed?**
- **How many transistors are needed? 20**

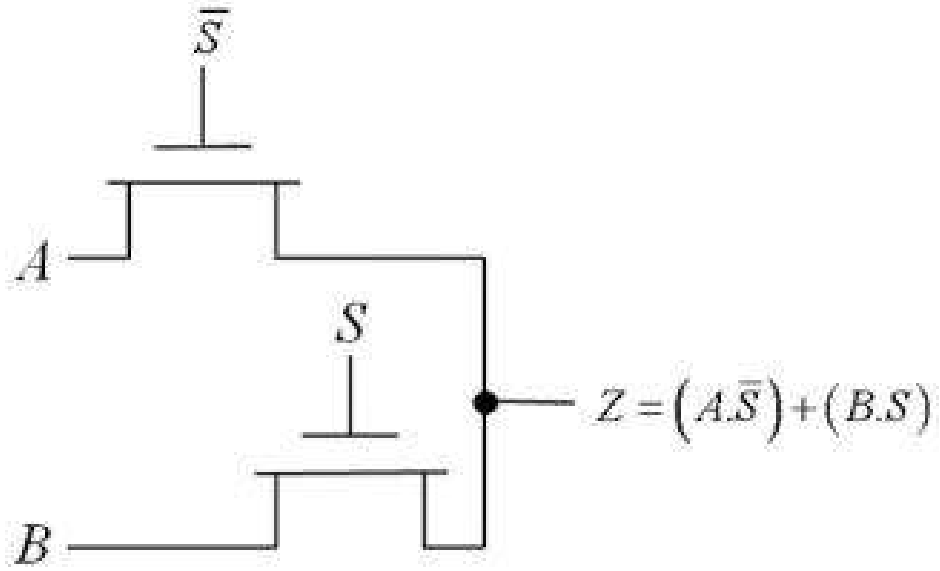


Structured Design-Mux Design-Transmission Gate

- Nonrestoring mux uses two transmission gates
  - Only 4 transistors



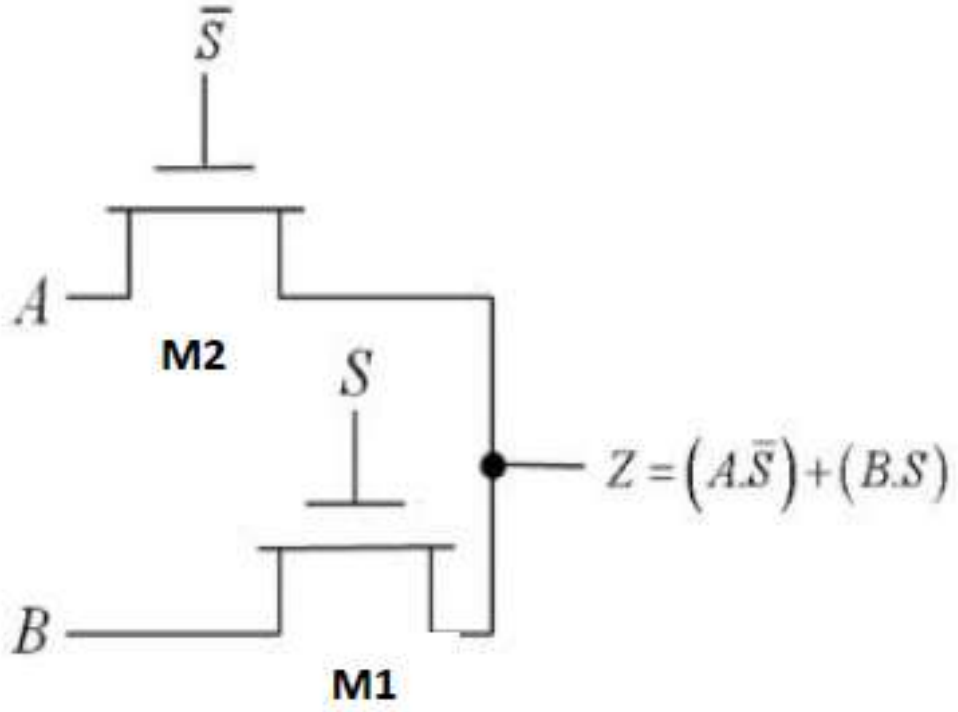
Design of 2:1 MUX using pass-transistor logic



- **The pass-transistor logic attempts to reduce the number of transistors to implement a logic by allowing the primary inputs to drive gate terminals as well as source-drain terminals.**

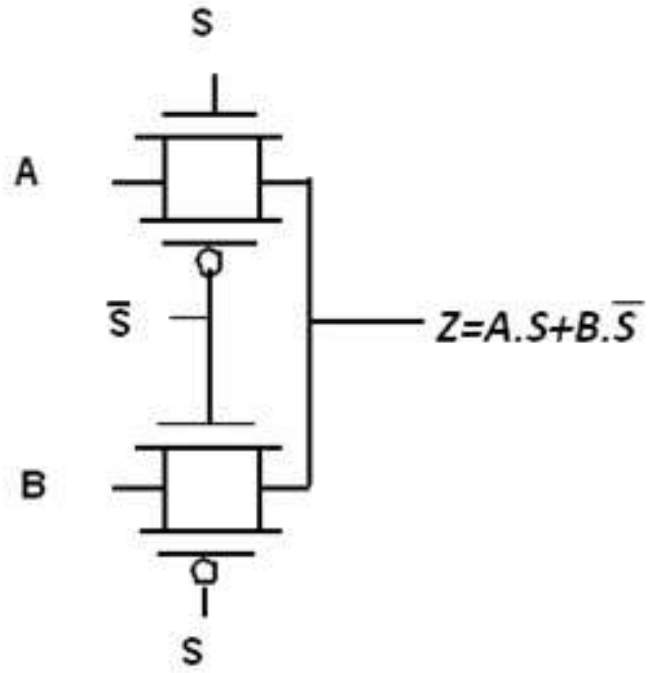
- **The implementation of a 2:1 MUX requires 4 transistors (including the inverter required to invert S), while a complementary CMOS implementation would require 6 transistors. The reduced number of devices has the additional advantage of lower capacitance.**

WORKING



- When S = 0, transistor M1 is OFF and M2 is ON, thus output Z = AS’
- When S = 1, transistor M1 is ON and M2 is OFF, thus output Z = BS

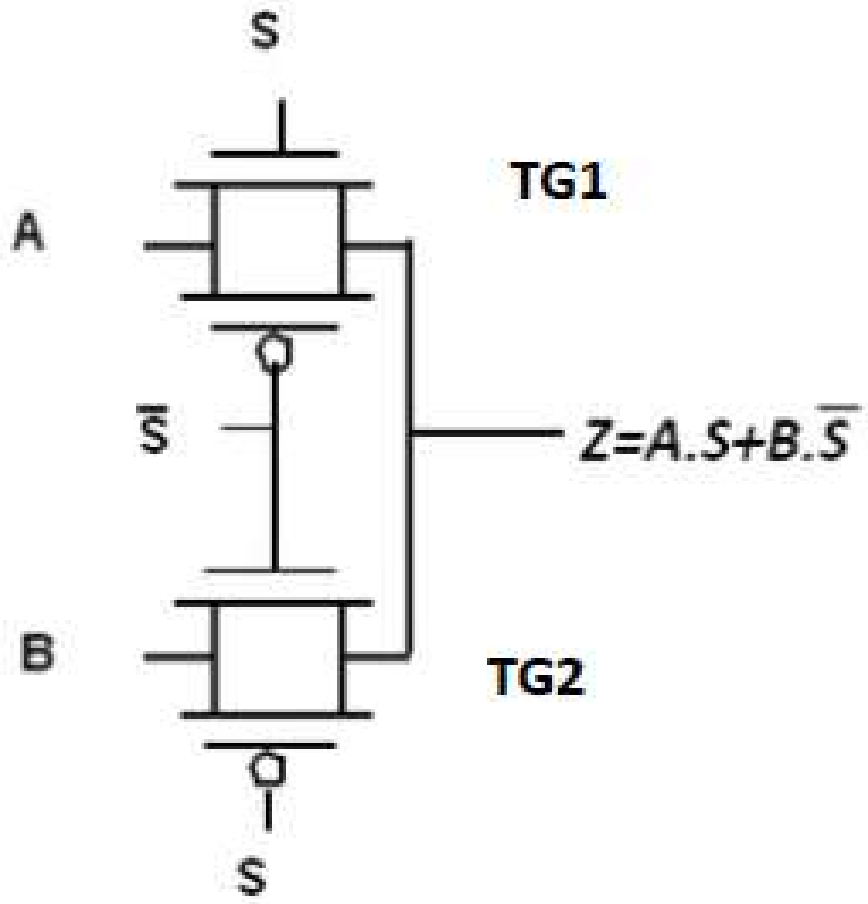
Design 2:1 MUX using transmission gate(TG) logic



The transmission gate acts as a bidirectional switch controlled by the gate signal C. When C=1, both MOSFETs are on, allowing the signal to pass through the gate. In short, A=B, if C=1. On the other hand, C=0, places both transistors in cut-off, creating an open circuit between nodes A and B.

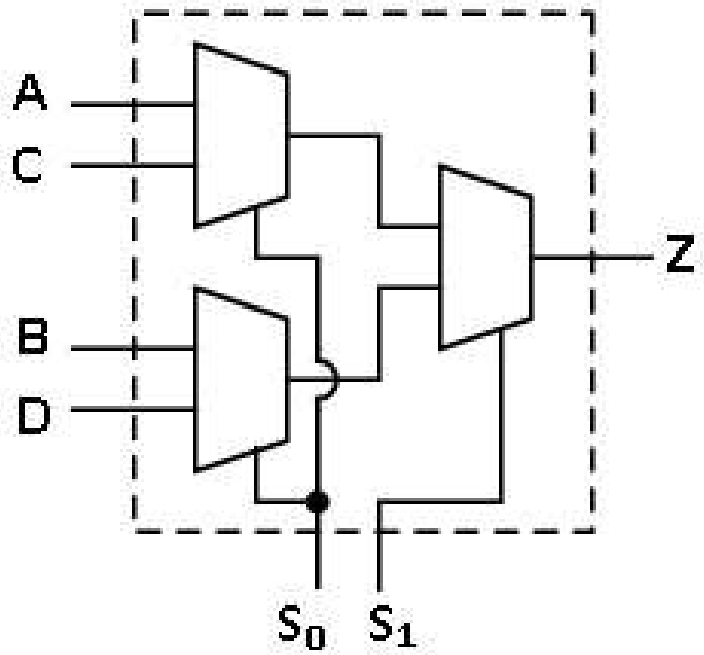
Here, the transmission gates selects input A or B on the basis of the value of the control signal S. When S=0, Z=A and when S=1, Z=B.

WORKING



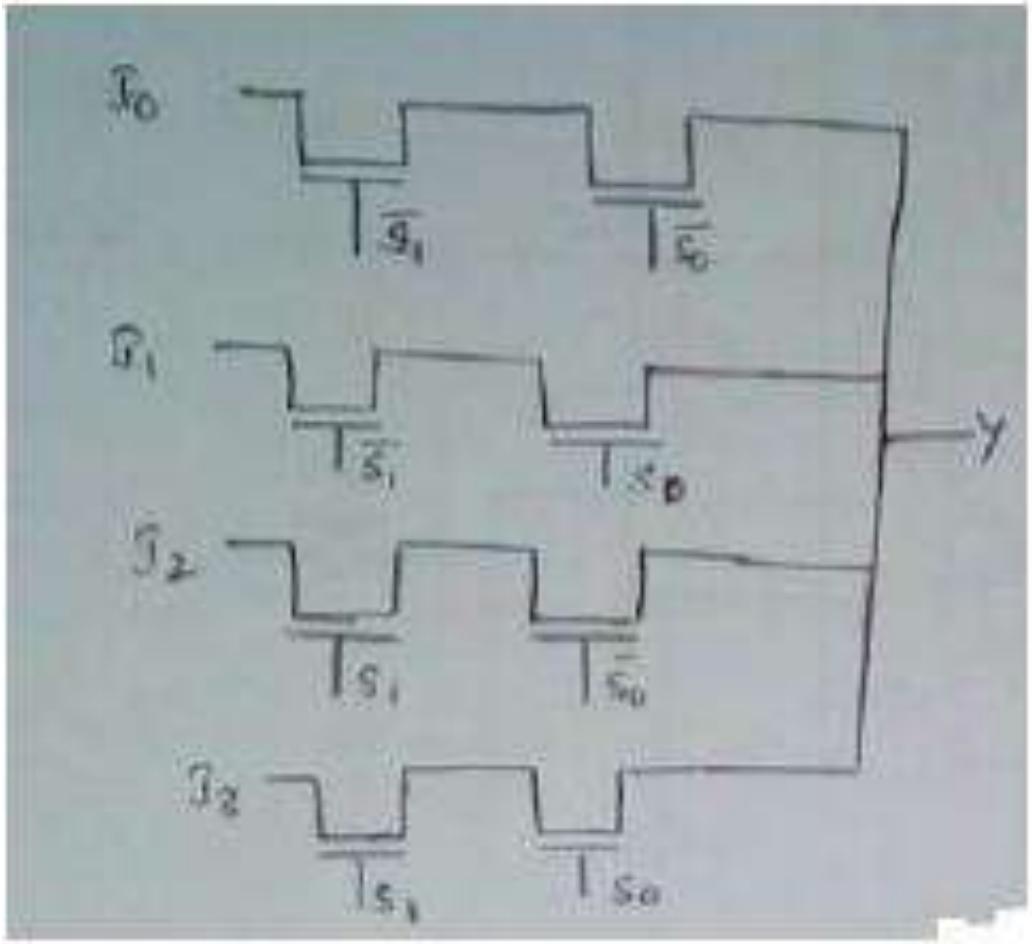
- The transmission gate acts as a bidirectional switch controlled by the gate signal
- When S = 0, TG1 = OFF and TG2 is ON and Z = B.
- When S = 1, TG1 = ON and TG2 is OFF and Z = A.

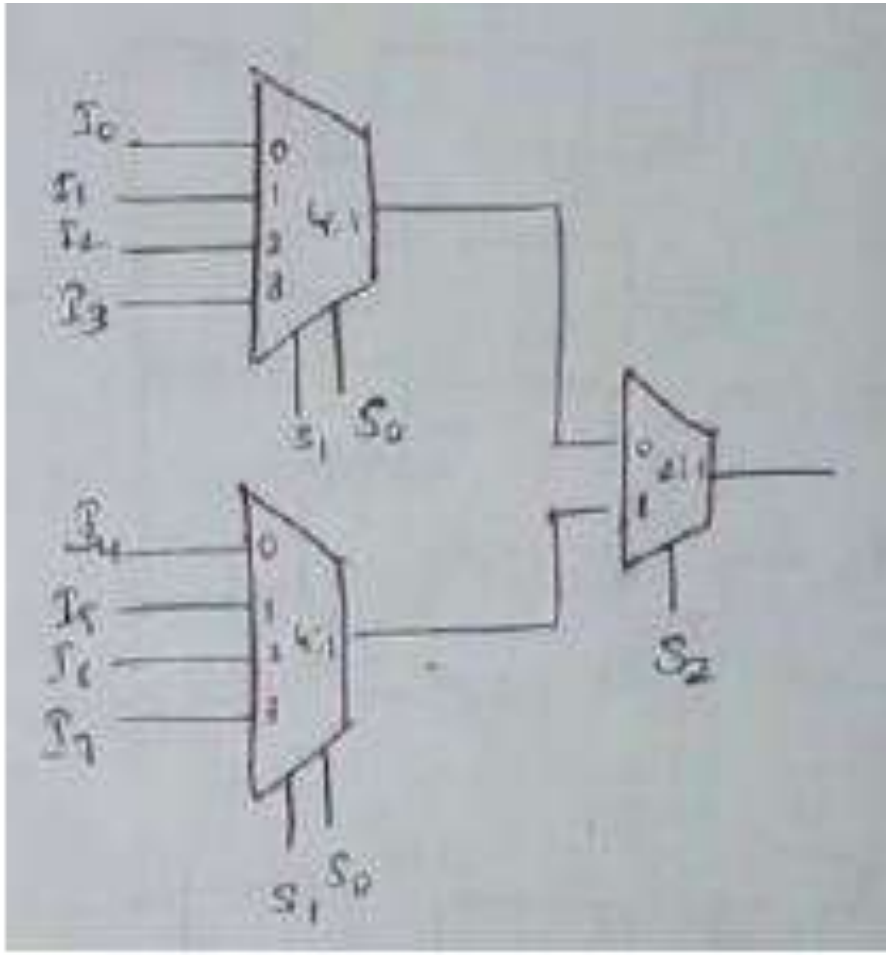
Implementation of 4:1 MUX using 2:1 MUXs



$$Z = \left(A.\overline{S}_0.\overline{S}_1\right) + \left(B.\overline{S}_0.S_1\right) + \left(C.S_0.\overline{S}_1\right) + \left(D.S_0.S_1\right)$$

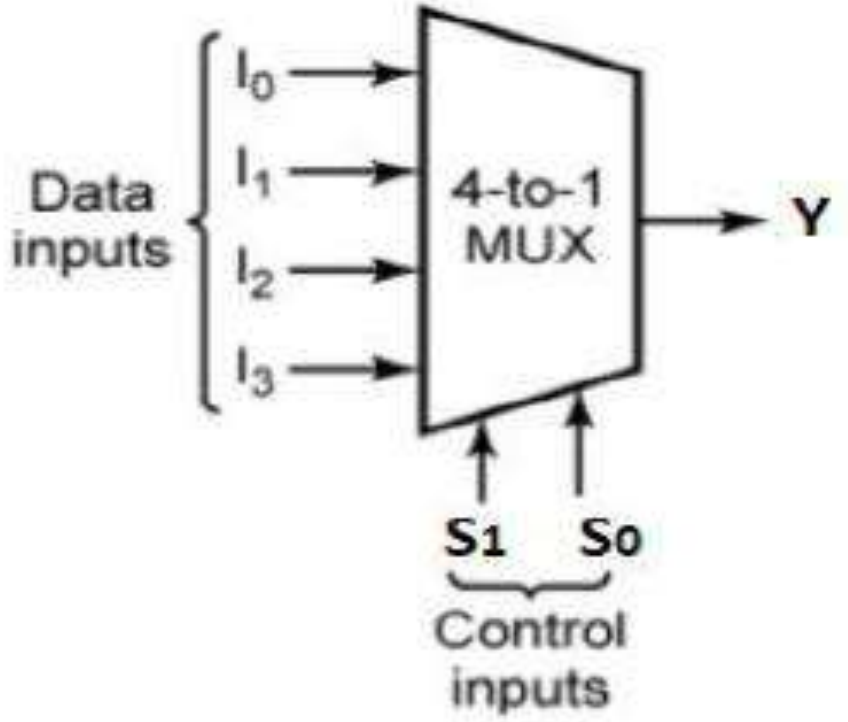
4:1 MUX implemented using two 2:1 multiplexer(Pass transistor Logic)





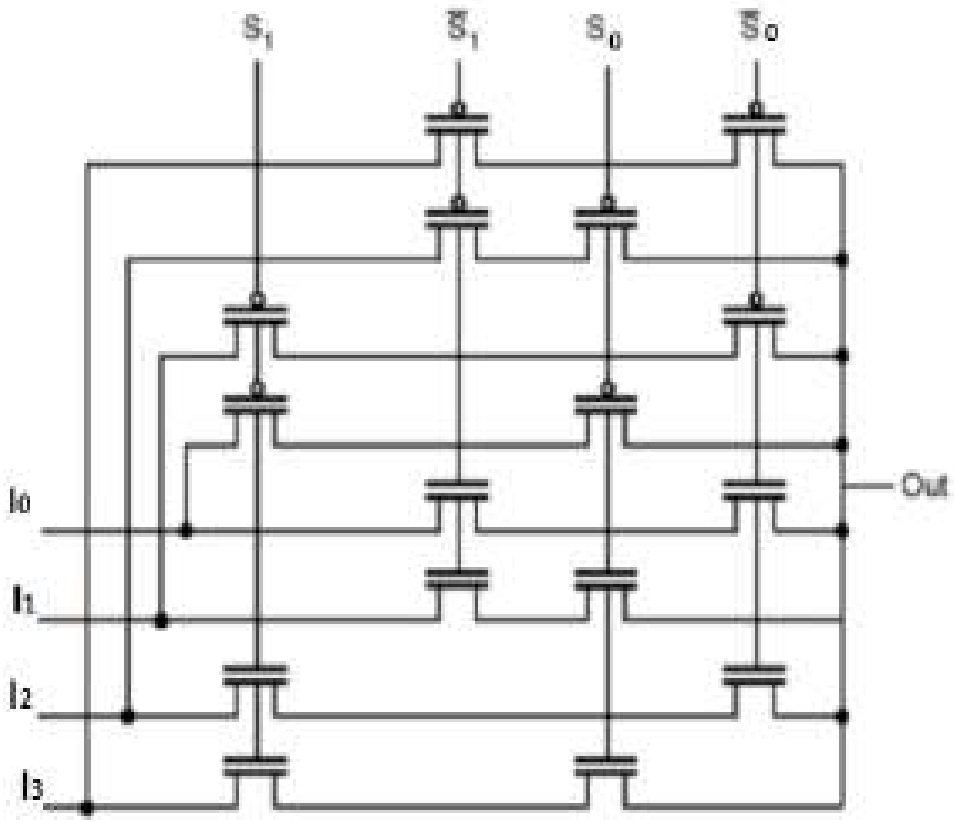
4:1 Multiplexer implementation(Four Way Crossbar Switch)

S1	S0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

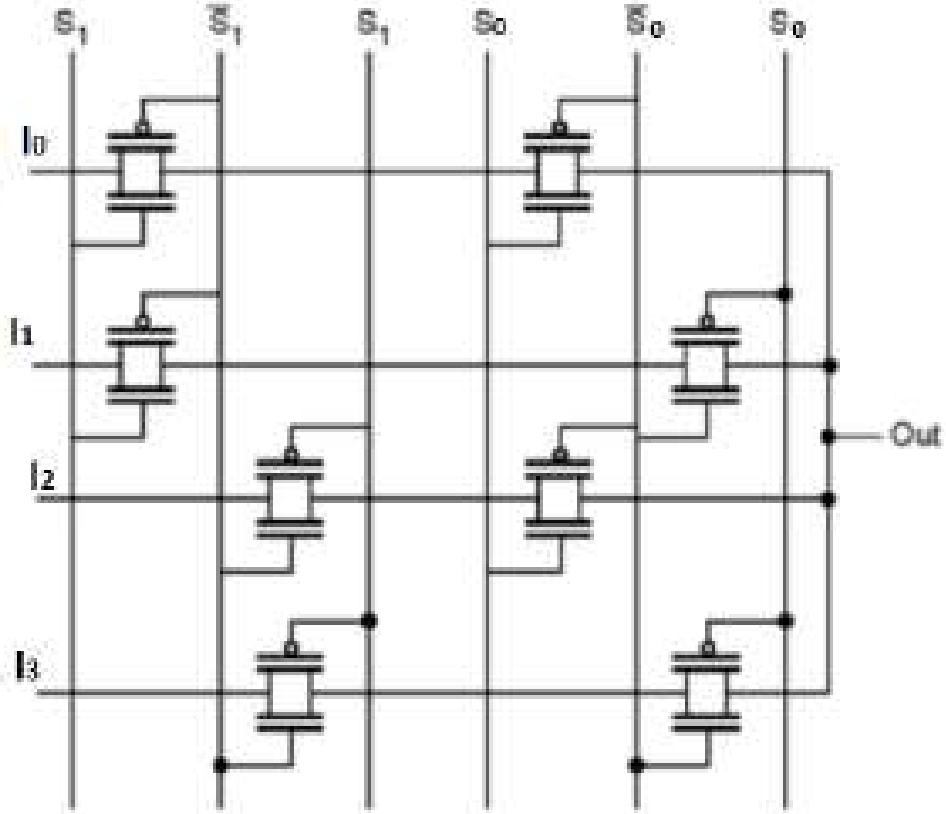


There are 4 inputs - I0, I1, I2, I3, 2 select lines S0, S1 and 1 output line – Y/out.

4:1 mux implementation using CMOS technology and transmission Gate (Four Way Crossbar Switch)

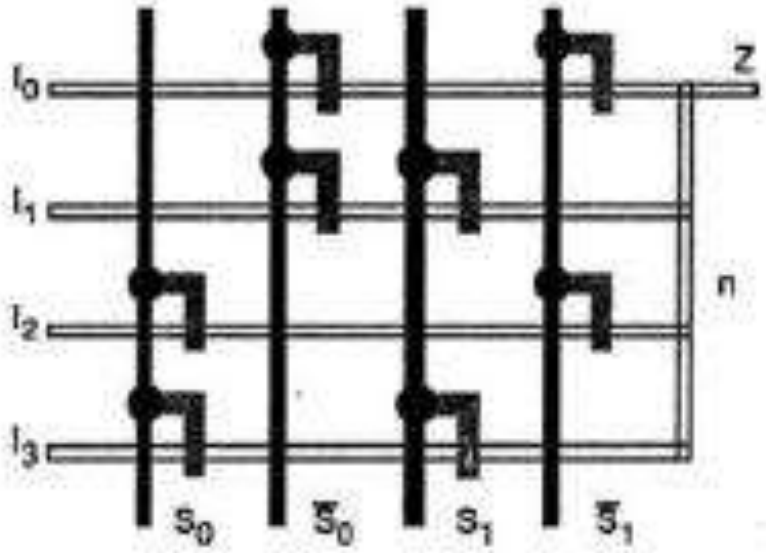


4 : 1 MUX using CMOS logic

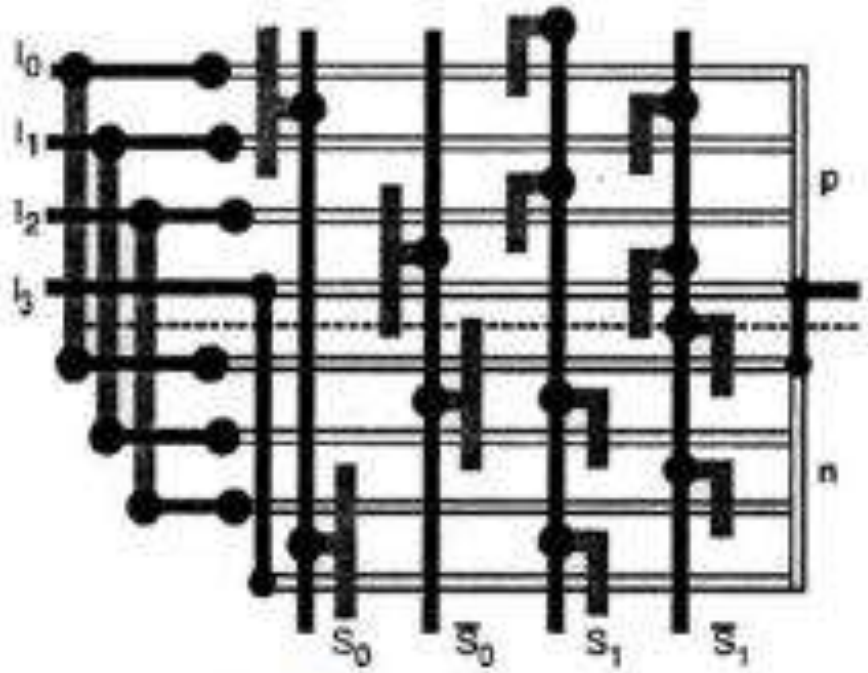


4 : 1 MUX using transmission gate

4:1 MUX Layout



(a) nMOS switches

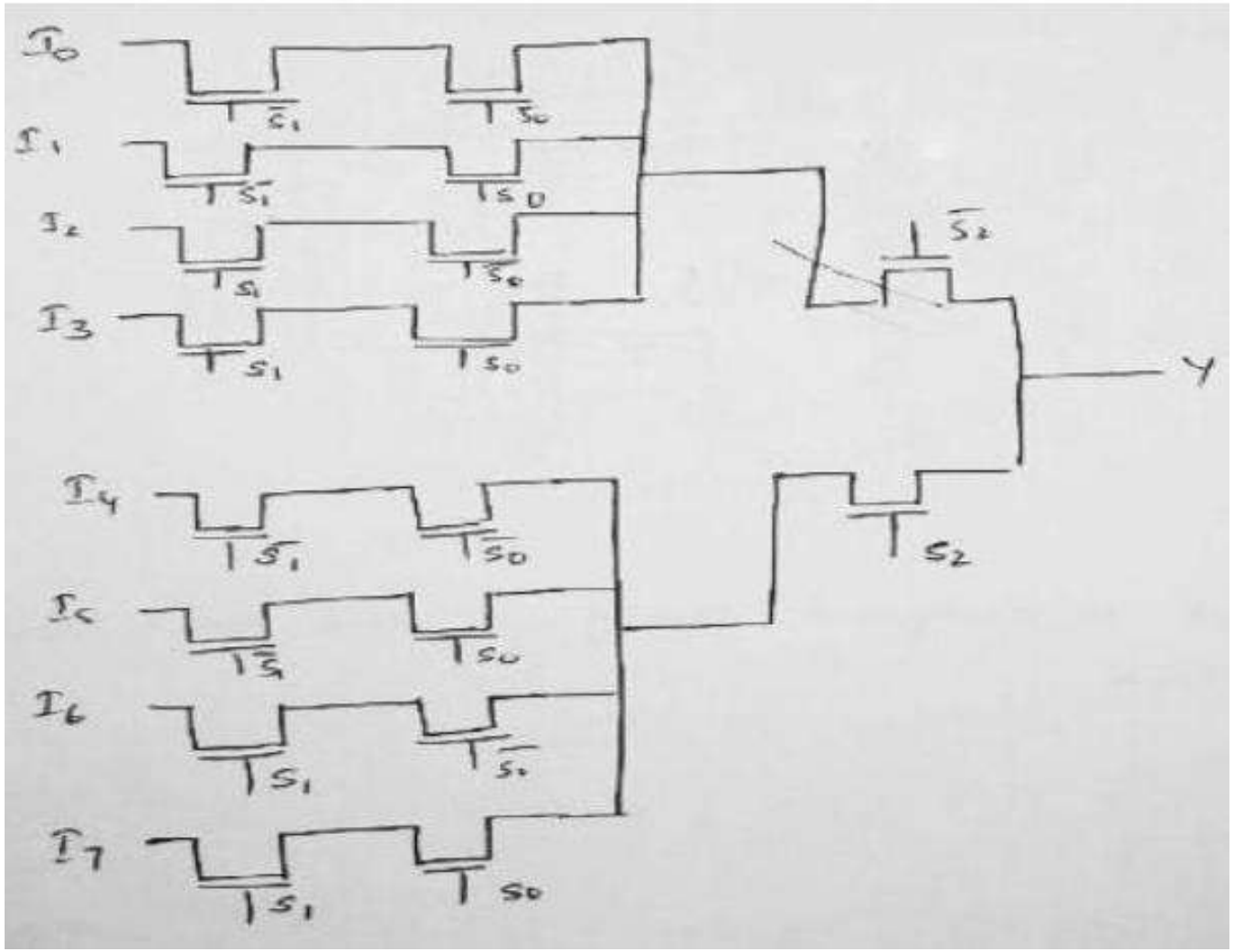


Note : V_{DD} and V_{SS} contacts are not shown.

(b) Transmission gates (CMOS)

Implementation of 8:1 using two 4:1 MUX

S2	S1	S0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7



Parity Generator

The parity generating technique is one of the most widely used error detection techniques for the data transmission.

- In digital systems, when binary data is transmitted and processed, data may be subjected to *noise.*
- Hence, **parity bit** is added to the word containing data in order to make number of 1s either even or odd. During the transmission of binary data, the message containing the data bits along with parity bit is transmitted from transmitter node to receiver node.
- At the receiving end, the number of 1s in the message is counted and if it *doesn’t match with the transmitted one, then it means there is an error in the data.*

- A parity generator is a combinational logic circuit that generates the parity bit in the transmitter. On the other hand, a circuit that checks the parity in the receiver is called parity checker. A combined circuit or devices of parity generators and parity checkers are commonly used in digital systems to detect the single bit errors in the transmitted data word.

- In even parity, the added parity bit will make the total number of 1s an even amount whereas in odd parity the added parity bit will make the total number of 1s odd amount.

- The basic principle involved in the implementation of parity circuits is that sum of odd number of 1s is always 1 and sum of even number of 1s is always zero. Such error detecting and correction can be implemented by using Ex-OR gates (since Ex-OR gate produce zero output when there are even number of inputs).

- Parity Generator: It is combinational circuit that accepts an n-1 bit stream data and generates the additional bit that is to be transmitted with the bit stream. This additional or extra bit is termed as a parity bit.

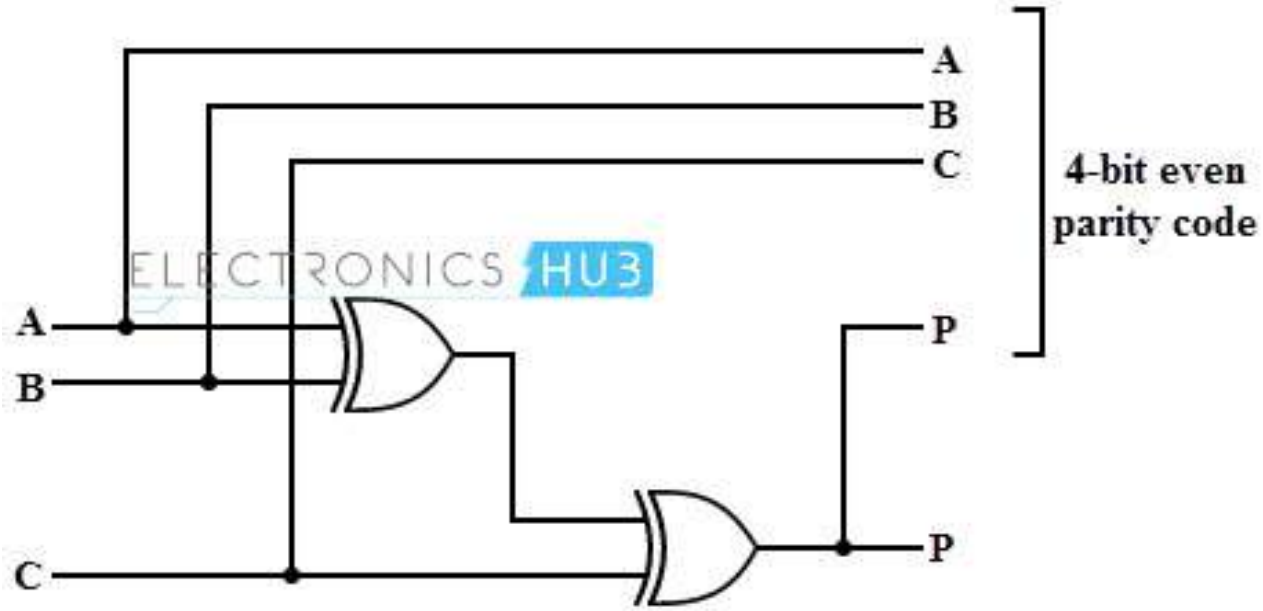
➢ **In even parity bit scheme, the parity bit is '0' if there are even number of 1s in the data stream and the parity bit is '1' if there are odd number of 1s in the data stream.**

➢ **In odd parity bit scheme, the parity bit is '1' if there are even number of 1s in the data stream and the parity bit is '0' if there are odd number of 1s in the data stream. Let us discuss both even and odd parity generators.**

Even Parity Generator

3-bit message			Even parity bit generator (P)
A	B	C	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

To generate the even parity bit for a 4-bit data, three Ex-OR gates are required to add the 4-bits and their sum will be the parity bit.



$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

$$= \bar{A}\,(\bar{B}\,C + B\,\bar{C}) + A\,(\bar{B}\,\bar{C} + B\,C)$$

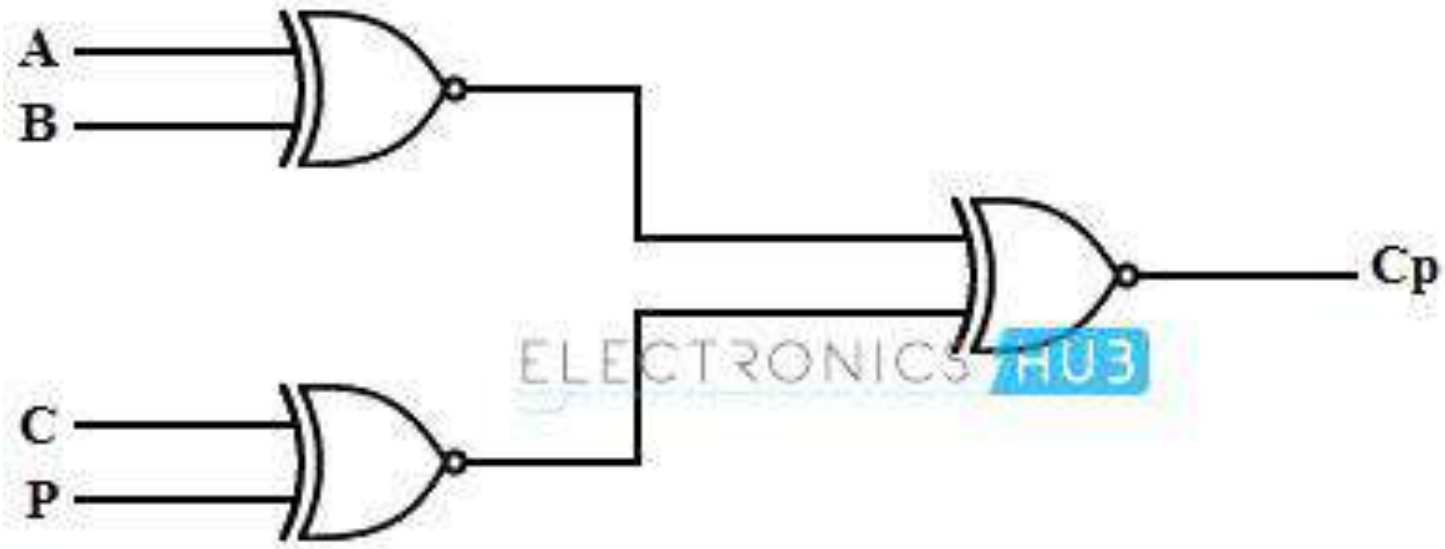
$$= \bar{A}\,(B \oplus C) + A\,\overline{(B \oplus C)}$$

$$P = A \oplus B \oplus C$$

Odd Parity Generator

3-bit message			Odd parity bit generator (P)
A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The logic circuit of this generator is shown in below figure , in which . two inputs are applied at one Ex-OR gate, and this Ex-OR output and third input is applied to the Ex-NOR gate , to produce the odd parity bit. It is also possible to design this circuit by using two Ex-OR gates and one NOT gate.



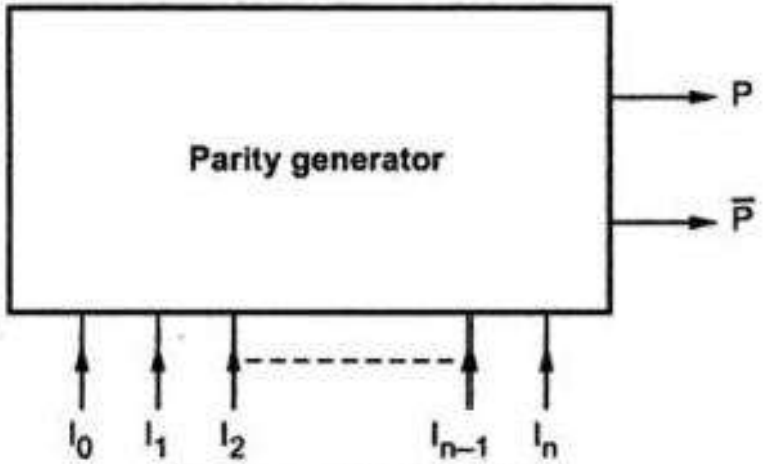
Even Parity Checker

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

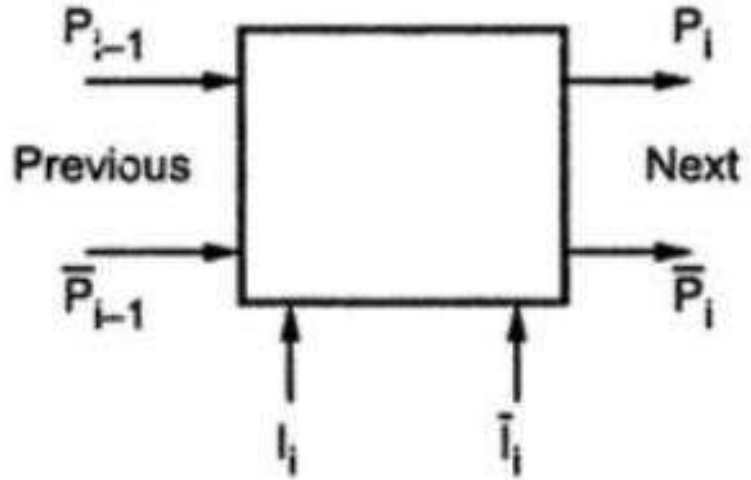
Odd Parity Checker

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

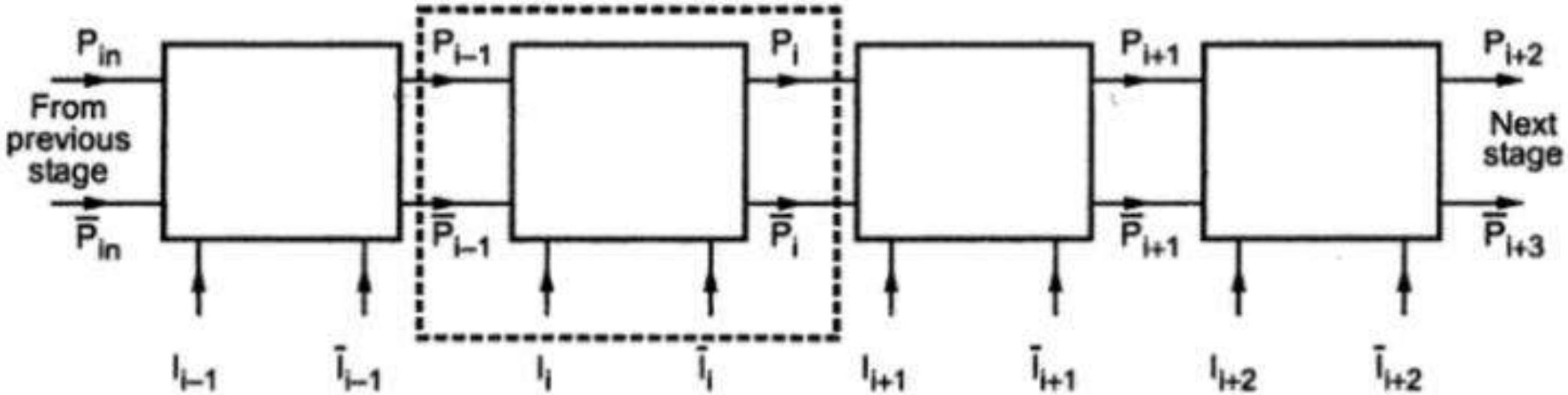
In VLSI a circuit to be designed to indicate parity of a binary number is shown in the Fig. for an (n+1) bit input.



Parity generator basic block diagram



Parity generator-basic one-bit cell



Note : Parity requirements are set at the left most cell where $P_{in} = 1$ sets even and $P_{in} = 0$ sets odd parity.

Fig. Parity generator - structured design approach

Since the no. of bits is undefined, a general solution on cascadable bit-wise and a regular structure is shown in Fig.

- A standard or basic one-bit cell from which an n-bit parity generator may be formed. The standard/ basic cell is shown in the Fig.

- The parity information is passed from one cell to next and the parity information is modified or retained depending on the input lines Ai and A’i

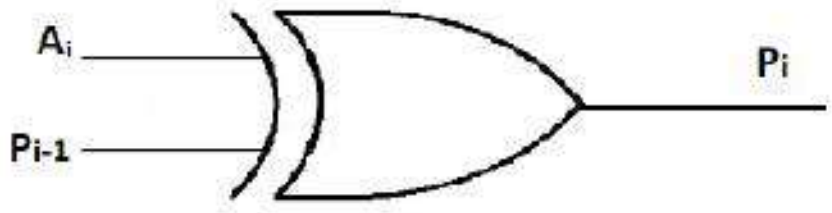
- If Ai = 1, parity output Pi will change to P'i-1 (i.e., if Ai [input] =1 and Pi-1 [previous parity] =1, the output parity Pi will change to 0)

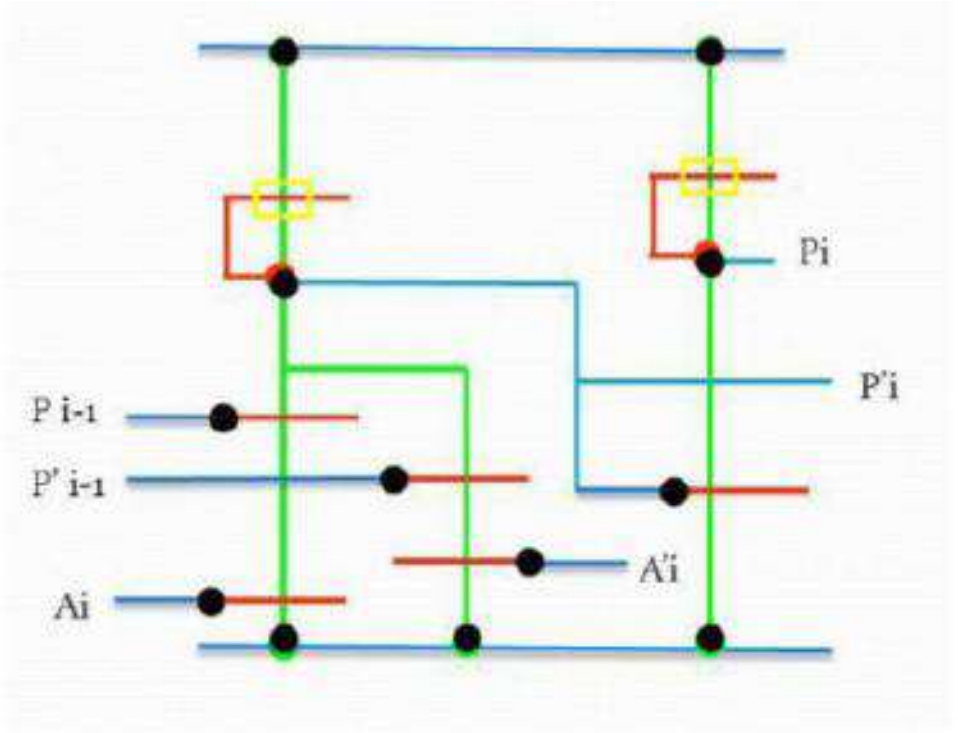
- If Ai = 0, output parity Pi will remain in the same state of Pi-1 (i.e., if Ai [input] =0 and Pi-1 [previous parity] =1, the output parity Pi will remain as 1)

- Suitable arrangement for such cell is implemented using the expression

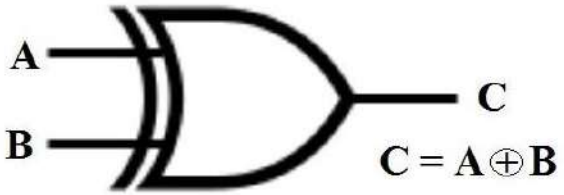
$$P_i = P'_{i-1} A_i + P_{i-1} A'_i$$

The parity generator symbol and stick diagram for nMOS technology is shown below.





XOR GATE Truth Table



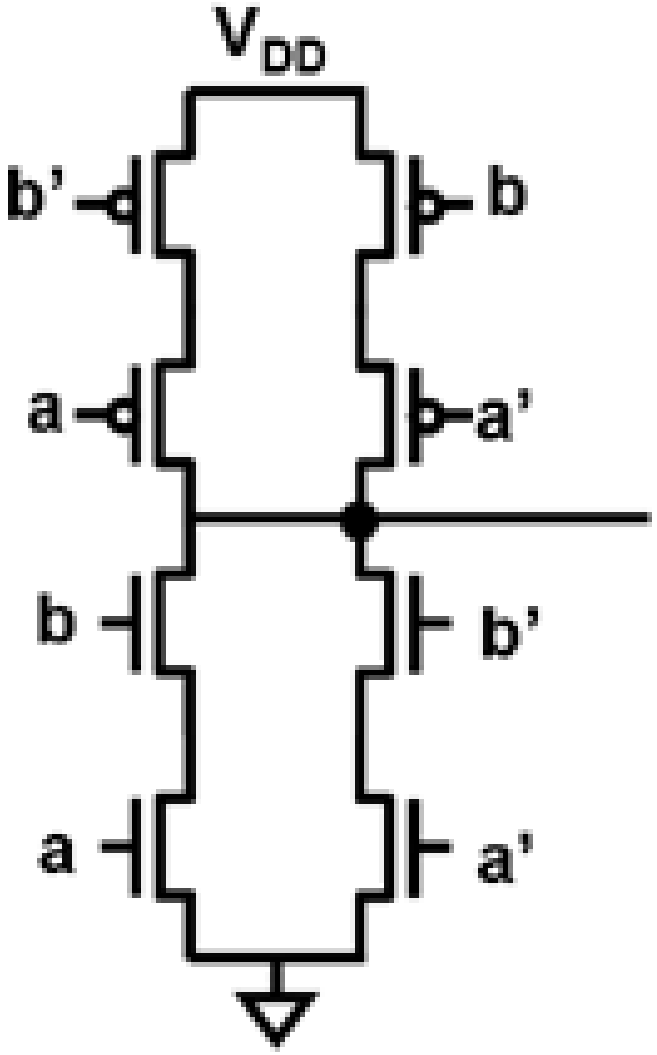
INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

BOOLEAN EXPRESSION

$$A \cdot \overline{B} + \overline{A} \cdot B$$

$$(A + B) \cdot (\overline{A} + \overline{B})$$

$C = A \oplus B$ (Input1, Input2, Output)

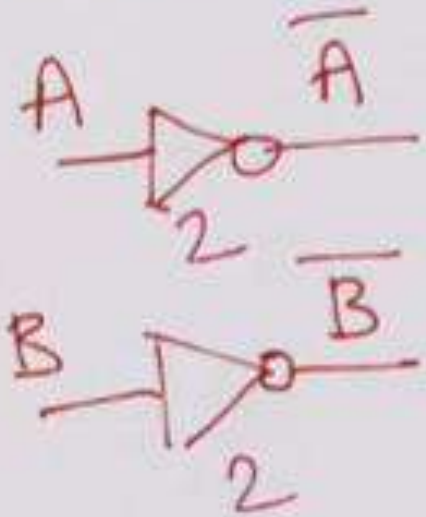


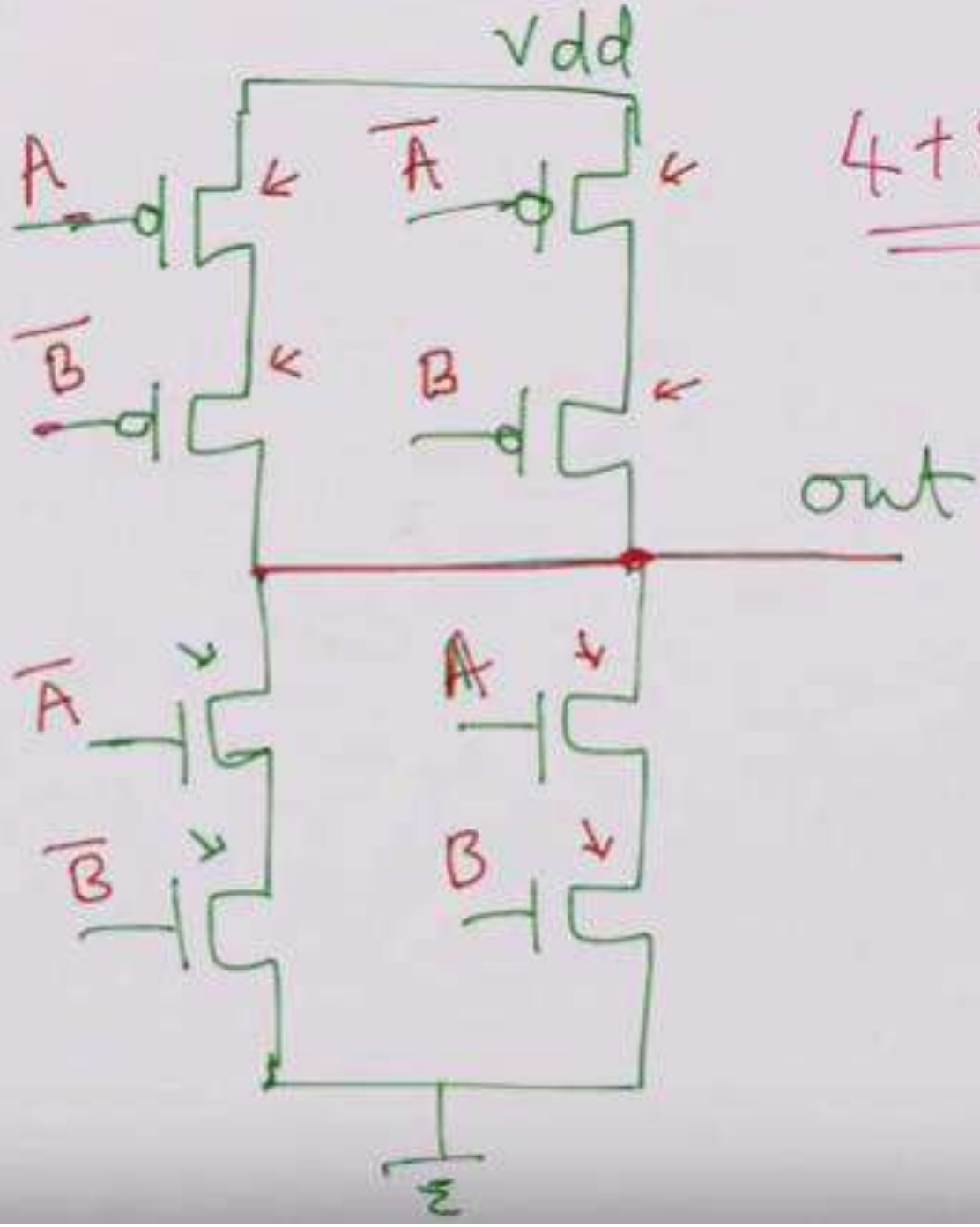
XOR

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

GND

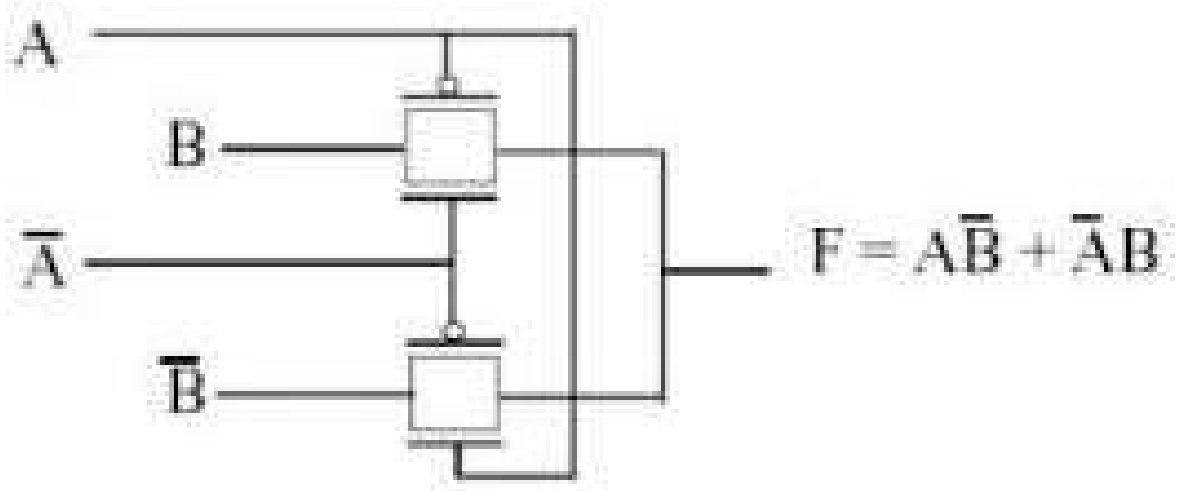
GND





4+8=12

XOR GATE USING TG



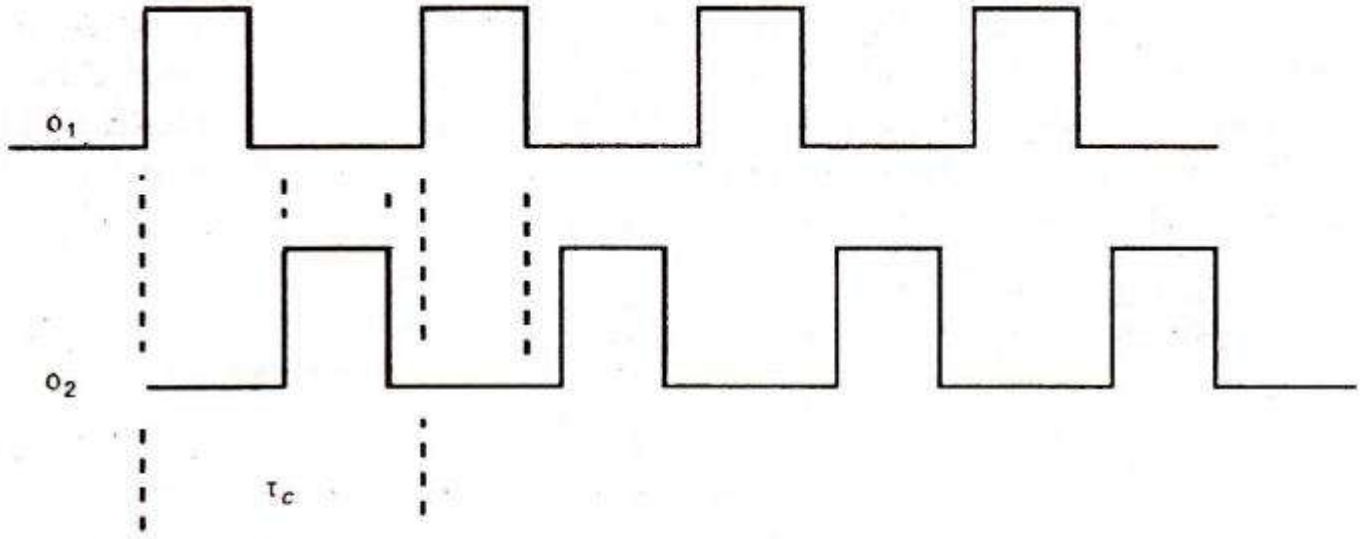
Examples: Structured Design(Top-down) Approach of clocked sequential circuits.

1. Two-phase clock generator using D flip-flops.
2. 4-bit Dynamic Shift Register.

Two-phase clock generator using D flip-flops.

Two-Phase Clocking:

The clocked circuits to be considered here will be based on a two-phase non-overlapping clock signal as defined by Figure.



Notes: 1. τ_c = clock period
2. $o_1(t)$. $o_2(t) = 0$; *all* t

FIGURE 6.31 Two-phase clocking.

- A two-phase clock offers a great deal of freedom in sequential circuit design if the clock period and the duration of the signals PHI-1 and PHI-2 are correctly chosen.

- If this is the case, data is allowed to become stable before any further transfer takes place and there is no chance of race conditions occurring.

- Clocked circuitry is considerably easier to design than the corresponding asynchronous sequential circuitry. It does, however, usually pay the penalty of being slower. However, at this stage of learning VLSI design we will concentrate on two-phase clocked sequential circuits alone and thus simplify design procedures.

- When studying Figure 6.31, it is necessary to recognize the fact that PHI-1 and PHI-2 do not need to be symmetrical as shown.

- For a given clock period, each clock phase period and its. associated under lap period can be varied if the need arises in optimizing a design.

- A number of techniques are used to generate the two clock phases. One popular method is illustrated in Figure 6.32 and it will be seen that the output frequency is one-quarter of that of the input clock.

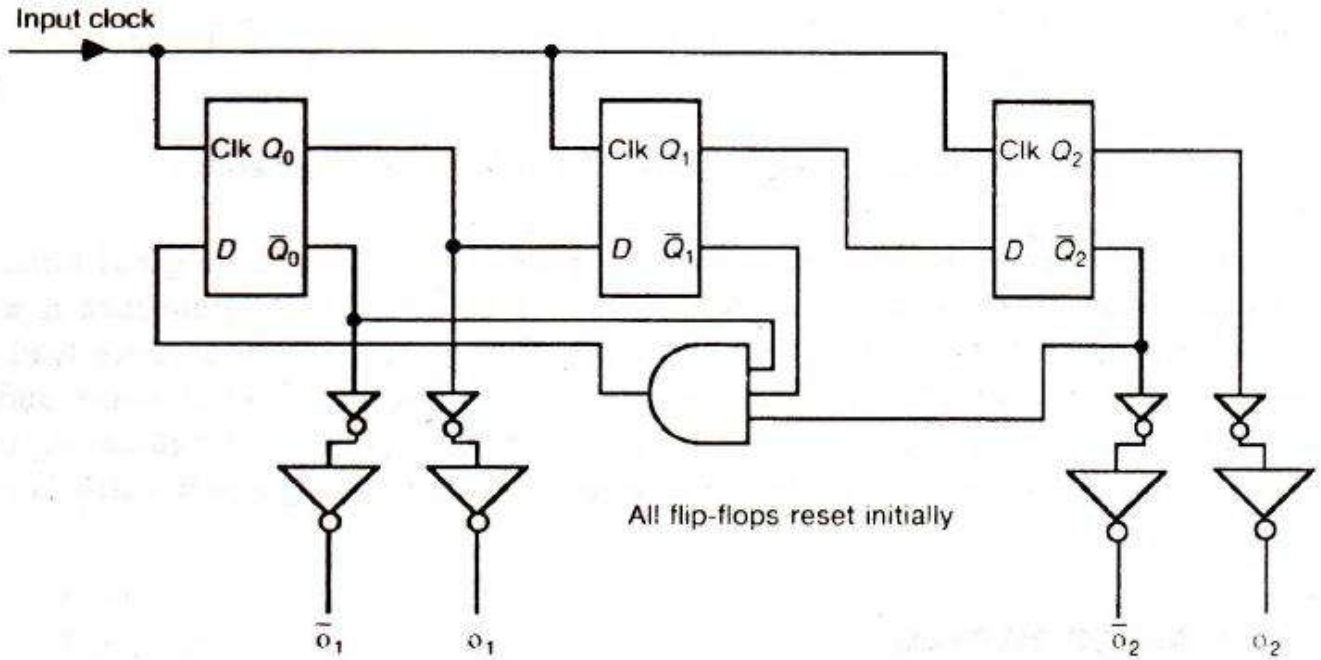


FIGURE 6.32 Two-phase clock generator using D flip-flops.

- A very simple arrangement using combinational logic and generating a two-phase clock at the frequency of a single-phase input clock is set out in Figure 6.33(a).

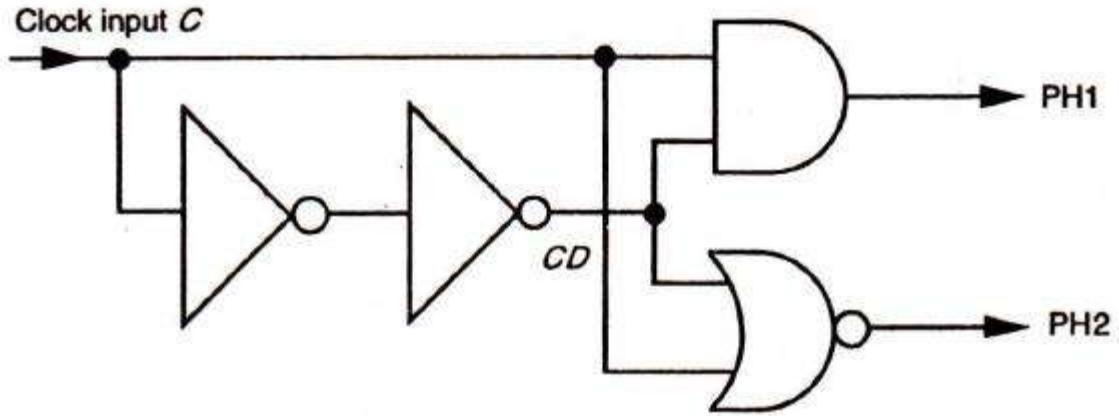


FIGURE 6.33(a) Simple two-phase clock generator circuit—basic form.

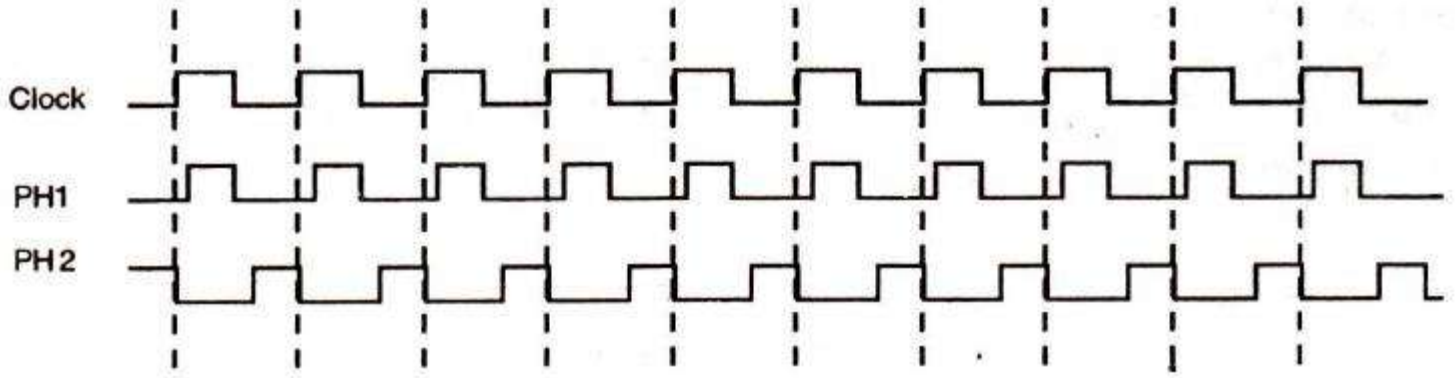


FIGURE 6.33(b) Waveforms for two-phase clock generator.

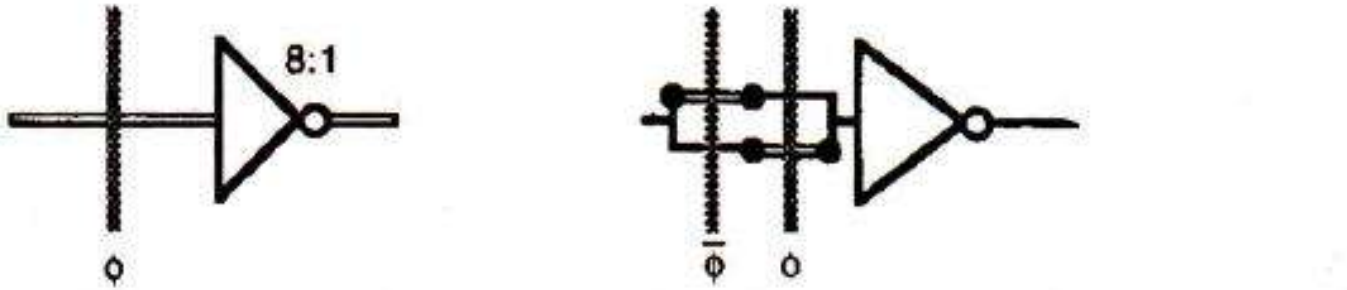
- The input clock signal C is used to provide a delayed version of itself (CD) by passing it through an even number of inverters. The delay thus produced determines the underlap period for the two phase clock.

- Waveforms are as shown in Figure 6.33(b). The phase 1 signal PHI-1 is generated by ANDing C with CD whilst the phase 2 signal PHI-2 is produced by NORing C with CD (that is, ANDing C' with CD'). Clearly, the minimum underlap period will be that generated by the delay through two inverters and this is also the increment by which the delay may be increased by adding further inverter pairs.

4-bit Dynamic Shift Register.

Dynamic Register Element

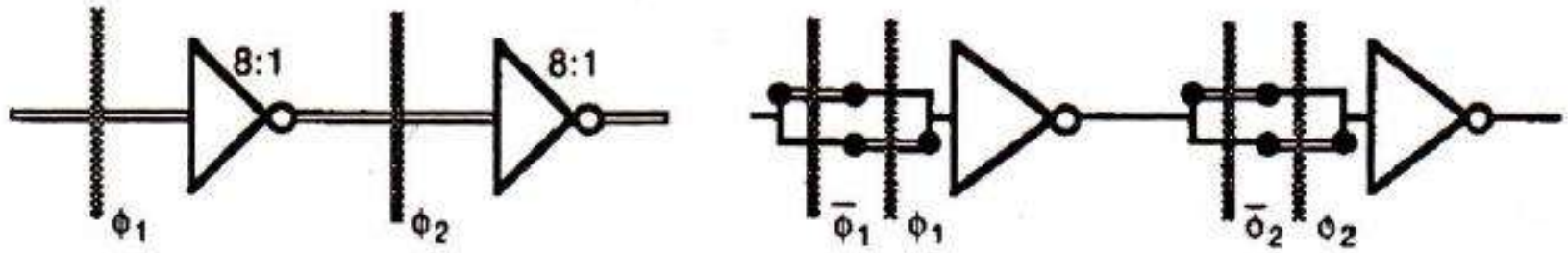
The basic dynamic register element is shown in Figure 6.36 in mixed stick/circuit notation and may be seen to consist of three transistors for nMOS and four for CMOS per stored bit in complemented form. The element's operation is simple to appreciate. (V_{in}) is clocked in by PHI-1 (or PHI-2) of the clock and charges the gate capacitance Cg of the inverter to V_{in}



(a) nMOS pass transistor switched (b) CMOS transmission gate switched

FIGURE 6.36 Basic inverting dynamic storage cells.

- If uncomplemented storage is essential, the basic element is modified as indicated in Figure 6.37 and will be seen to consist of six transistors for nMOS and eight for CMOS. Data clocked in on PHI-1 is stored on Cg1 and the corresponding output appears at the output of inverter 1. On PHI-2 this value is clocked into and stored by Cg2 and the output of inverter 2 then presents the 'true' form of the stored bit. Note that data read in on PHI-1 is not available at the output until sometime following the next positive edge of the clock signal PHI-2.



(a) nMOS pass transistor switched (b) CMOS transmission gate switched

FIGURE 6.37 Non-inverting dynamic storage cells.

- ***Dynamic storage elements and the corresponding register arrays are used in situationswhere signals are updated frequently (i.e. at < 0.25 msec intervals).***

4-bit Dynamic Shift Register

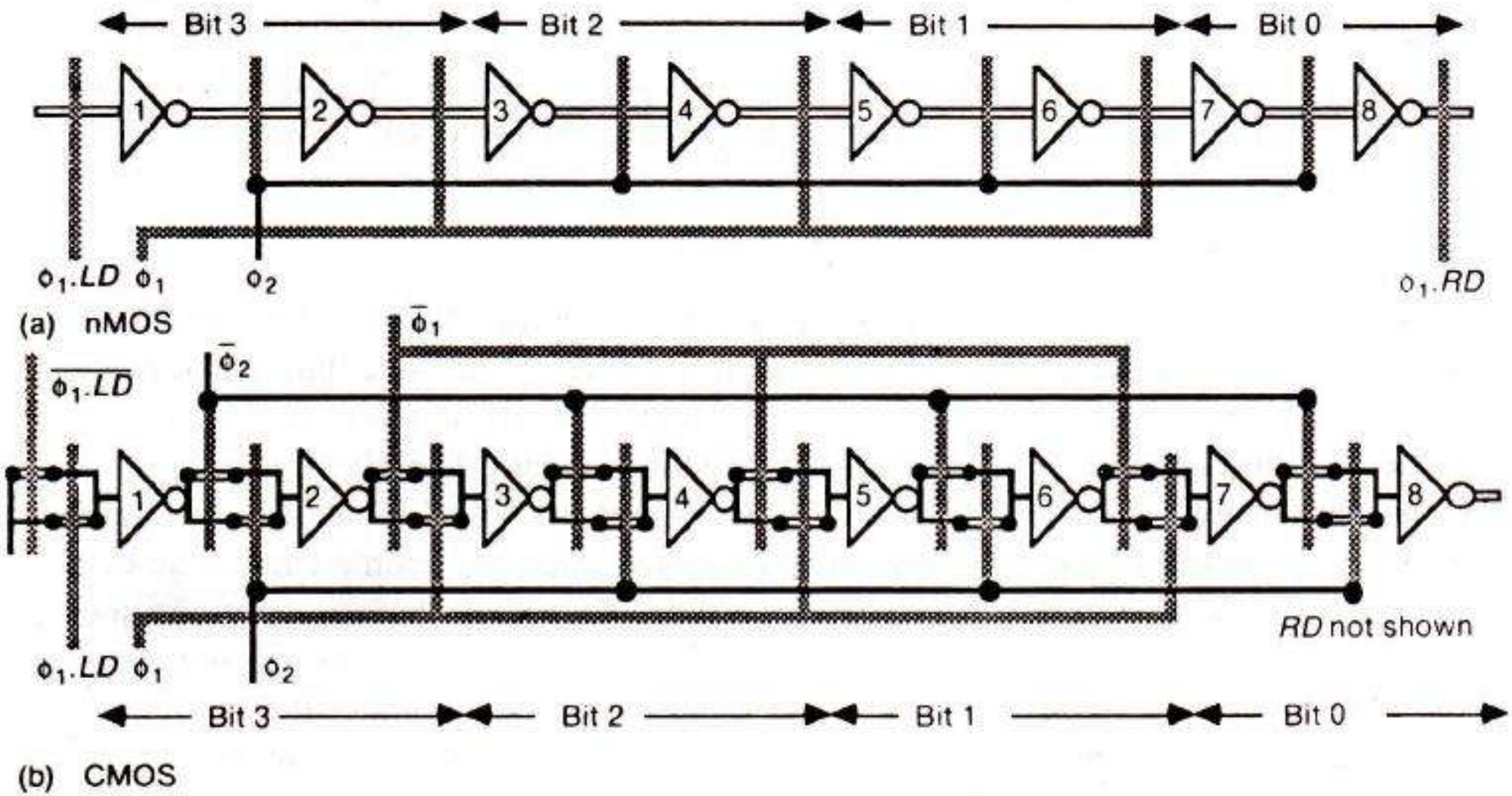


FIGURE 6.38 Four-bit dynamic shift registers (nMOS and CMOS).

- Cascading the basic elements of Figure 6.37 gives a serial shift register arrangement which may be extended to *n* bits. A four-bit serial right shift nMOS register is illustrated in Figure 6.38(a).

- Data bits are shifted in when ¢1.LD is present, one bit being entered on each ¢1signal (provided that *LD* is logic 1). Each bit is stored in Cg1 as it is entered, and then transferred complemented into *Cg2* during the next ¢2. Thus, after a ¢1 followed by ¢2 signal, the stored bit is present at the output of inverter 2.

- On the next ¢1 the next input bit is storedin Cg1 and simultaneously the first bit stored is passed on to inverter pair 3 and 4 by being stored in *Cg3,* and so on.

- It will be seen that bits are thus clocked to the right along the shift register on each ¢1 followed by ¢2 sequence.

- Once four bits are stored, the data is available in parallel form at the outputs of inverters 2, 4, 6 and 8, and is also available in serial form from the output of inverter 8 when ¢ii.RD is high as further clock sequences are received (where *RD* is the serial read control signal).

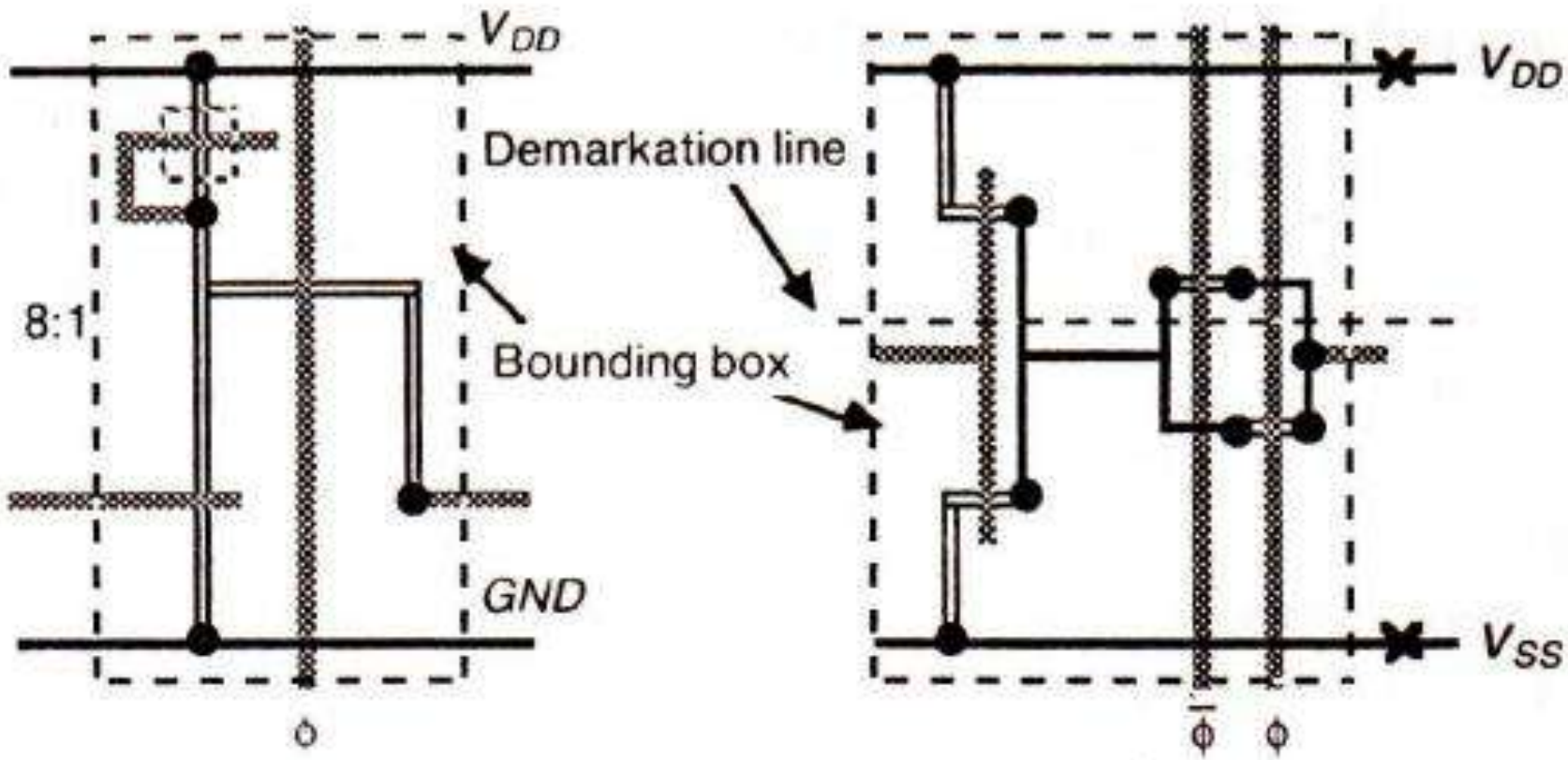


FIGURE 6.39 Stick diagrams for shift register cells.

Reliability and Testing of VLSI Circuits

Reliability and Testing of VLSI Circuits

- After the chip is fabricated it is tested for manufacturing defects. The chip designer must verify or validate the design to ensure the circuit performance. *Verification or validation is a different process than testing.*

- Verification is related to formal proof of correctness. While validation is a technique that increases confidence in correctness.

- *Testing is used not only to find the fault-free devices, PCBs and systems but also to improve production yield* at the various stages of manufacturing by analyzing the cause of defects when faults are encountered. In some systems, periodic testing is performed to ensure fault-free system operation.

"If you don't test it, it won't work ! (Guaranteed)."

Difference between Verification and Testing

- Verification guarantees the correctness of the design.
- Performed *once* before the actual manufacturing of the circuits / chips.
- Primarily responsible for the quality of the design.
- Uses formal methods, simulation, etc.

- Testing *tries to* guarantee the correctness of the manufactured circuits / chips.
- Has to be performed on *every* manufactured device.
- Primary responsible for the quality of the devices that go to the market.
- Two steps involved: (a) Test Generation, (b) Test Application.

- Test generation: software process executed once during design.
- Test application: electrical tests applied to hardware.

NEED OF TESTING CIRCUIT IN VLSI

- Possibility of errors during the design process i.e. to determine the presence of fault in given circuit/chip.
- There can even be bugs in the translation process (viz. the CAD tools used).
- Possibility of faults during fabrication / packaging.
- Necessary to test each and every chip before they can be used.
- Billions of transistors is present-day VLSI chips.
  — Chances of faults creeping in is also quite significant.

What are the Sources of Faults?

- Because of errors during the fabrication process.
  - Missing contact window, parasitic transistors, etc.
- Because of defects in the material(s).
  - Cracks or imperfections in the substrate, surface impurities, etc.
- Because of ageing.
  - Dielectric breakdown, electron migration, etc.
- Because of defects during packaging.
  - Contact degradation, disconnection, etc.

When to do Testing?

Can be carried out at various levels:

— At the chip-level, when chips are manufactured,

— At the board-level, when chips are integrated on the boards,

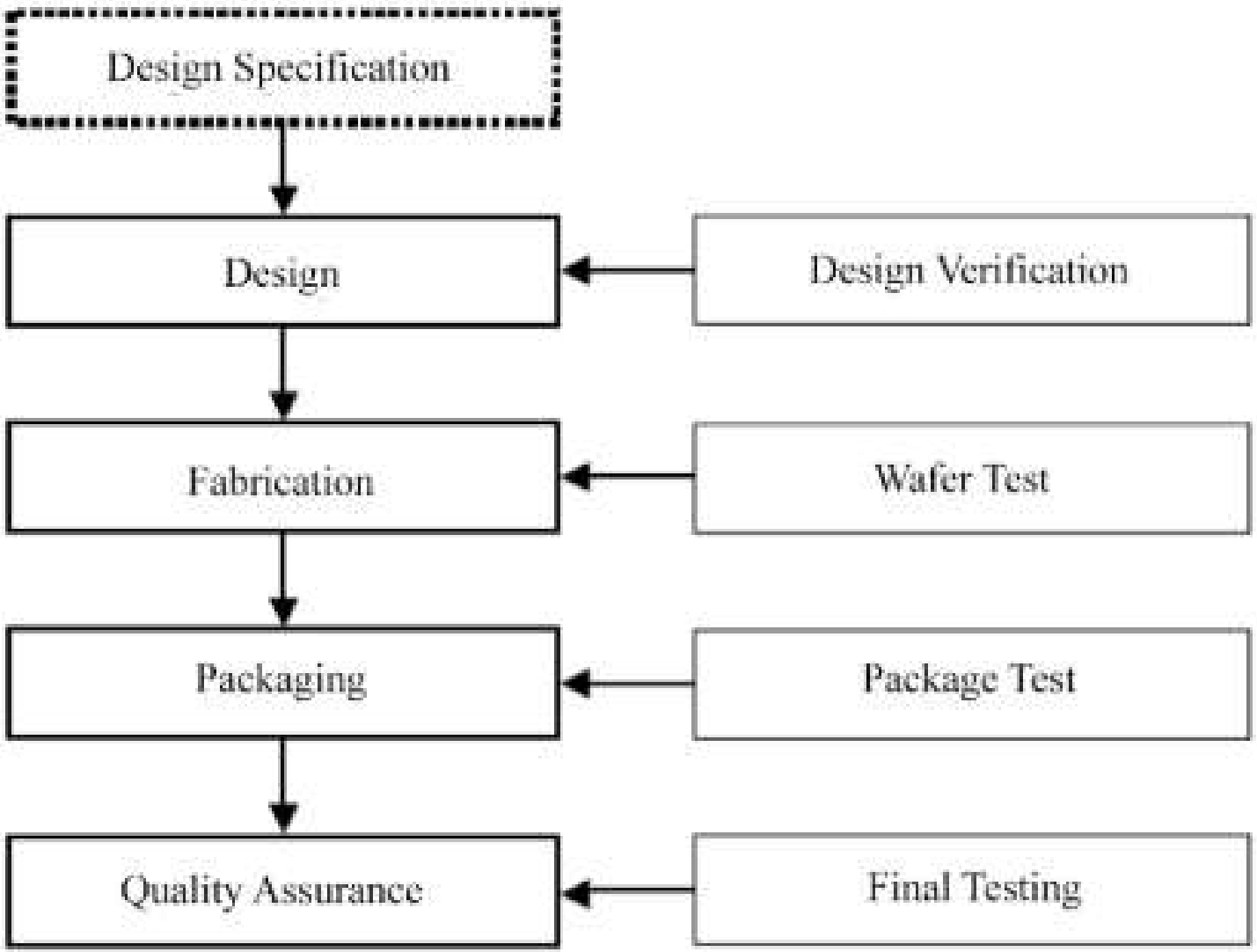
— At the system-level, when several boards are assembled together.

Rule of thumb:

— Detecting a fault early reduces the cost of testing*

— Empirical Rule-. It is 10 times more expensive to test a device as we move to the next higher level (chip —> board —> system).

Testing at Various Stages



Why Testing is Considered Difficult?

- Consider a combinational circuit with N inputs. Suppose we use a naïve way of testing where we apply all 2^N inputs and verify the truth table.

N inputs	= 2^N *combinations* =	$2^N \times 0.1$, *µsec.* =	*total test time*
32 inputs	2^{32}	$2^{32} \times 10^{-7}$ sec	≥ 7 minutes
40 inputs	2^{40}	$2^{40} \times 10^{-7}$ sec	≥ 30 hours
64 inputs	2^{64}	$2^{64} \times 10^{-7}$ sec	≥ 574 *centuries*

- Such naïve approach is not feasible as circuits increase in size.
  - Need some mechanism to enhance the controllability and observability of the state variables.
  - *Design for Testability* is a standard option that is used in practice.

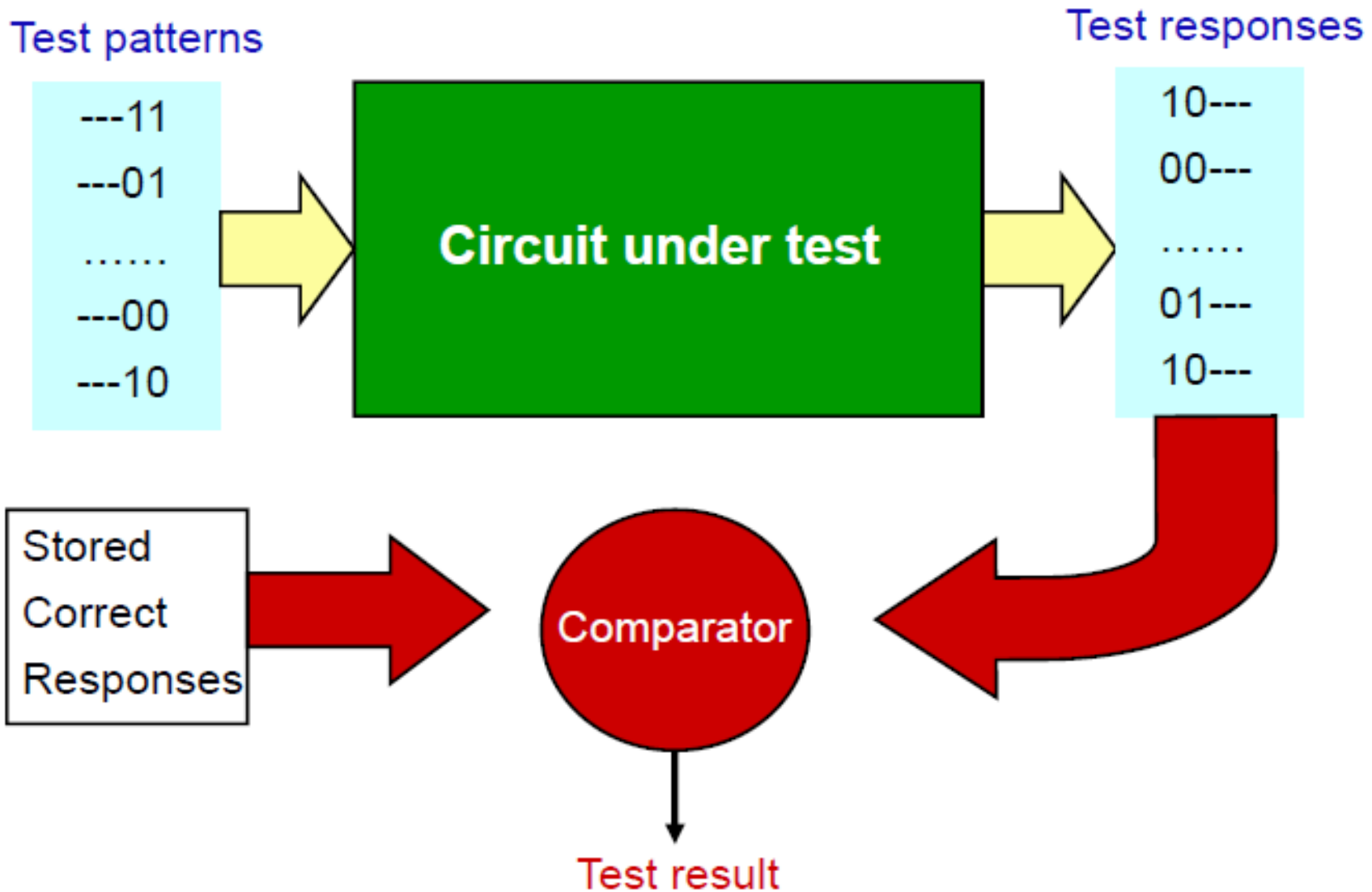
- Design for Testability (DFT)
  - Formulate a set of design rules that, if followed, results in a circuit that will be *easily testable*.
  - Typically introduces both area overhead and performance degradation.
- Built-in Self-Test (BIST)
  - Test generation and response evaluation of the circuits are performed on-chip.
  - The chip can test itself.
  - Additional area overhead.

Advantages of BIST

- Test patterns generated on-chip -controllability Increased
- Test can be on-line (concurrent) or off-line
- Test can run at circuit speed, more realistic; shorter test time;easier delay testing
- External test equipment greatly simplified, or even totally eliminated
- Easily adopting to engineering changes

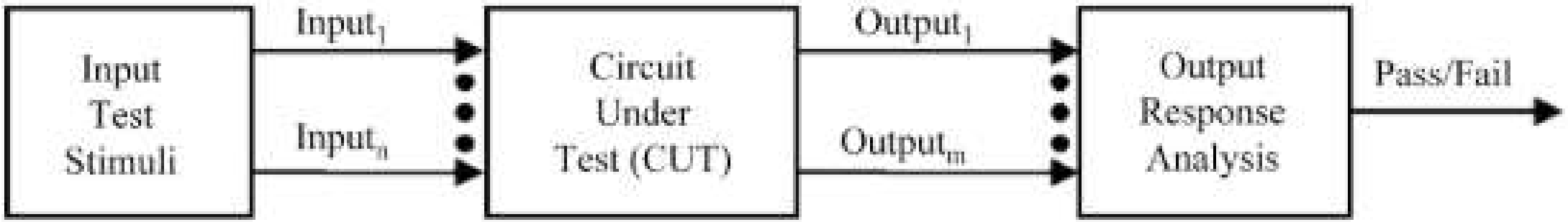
- Design for testability (DFT) refers to those design techniques that make test generation and test application cost-effective

How to Test Chips?



TESTING DURING THE VLSI LIFE CYCLE

- Testing typically consists of applying a set of test stimuli to the inputs of the circuit or device under test (CUT or DUT) while analyzing the output responses, as illustrated in Fig.



- Circuits that produce the correct output responses for all input stimuli pass the test and arc considered to be fault-free ,

- Those circuits that fail to produce a correct response al any point during the test sequence are assumed to be faulty.

- Testing is performed at various stages in the life cycle of a VLSI device. including during the VLSI development process, the electronic system manufacturing process, and. in some cases. system-level operation.

- A testable circuit is defined as a circuit whose internal nodes of interest can be set to 0 or 1 and in which any change to the desired logic value at the node of interest, due to a fault, can be observed extrenally.

- Based on a customer or project need, a VLSI device requirement is determined and formulated as a design specification.

- Designers are then responsible for synthesizing a circuit that satisfies the design specification and for verifying the design. Design verification is a predictive analysis that ensures that the synthesized design will perform the required functions when manufactured.

- When a design error is found, modifications to the design arc necessary and design verification must be repealed. As a result, design verification can be considered as a form of testing.

Testing Techniques

- Types of Testing Circuits
- Combinational Circuit Testing
  - Fault Model
  - Path Sensitizing
  - Random Test
- Sequential Circuit Testing
  - Scan Path Test
  - Built-in Self Test (BIST)
    - Built-in Logic Block Observer (BIBLO)
    - Signature Analyzer
  - Boundary Scan Test (BST)

****