VIGNANA BHARATHI
Institute of Technology

NATIONAL BOARD OF ACCREDITATION

NAAC
NATIONAL ASSESSMENT AND ACCREDITATION COUNCIL

Counselling Code
VBIT

VLSI

UNIT - V

Programmable Logic Devices

P.VIDYA SAGAR (ASSOCIATE PROFESSOR)

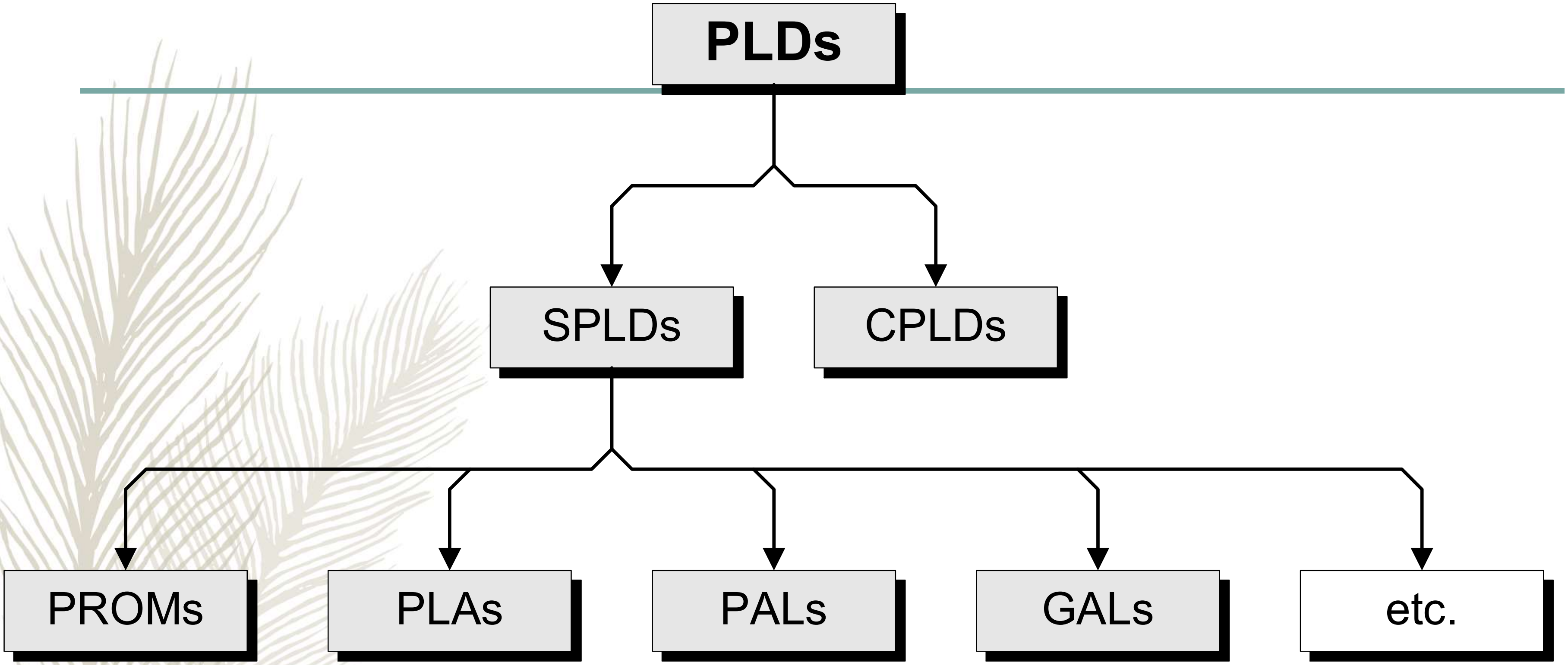
CONTENTS

SEMICONDUCTOR INTEGRATED CIRCUIT DESIGN: PLAs, FPGAs, CPLDs, Standard Cells, Programmable Array Logic, Design Approach, Parameters influencing low power design.

CMOS TESTING: CMOS Testing, Need for testing, Test Principles, Design Strategies for test, Chip level Test Techniques.

Programmable Logic Devices (PLD)

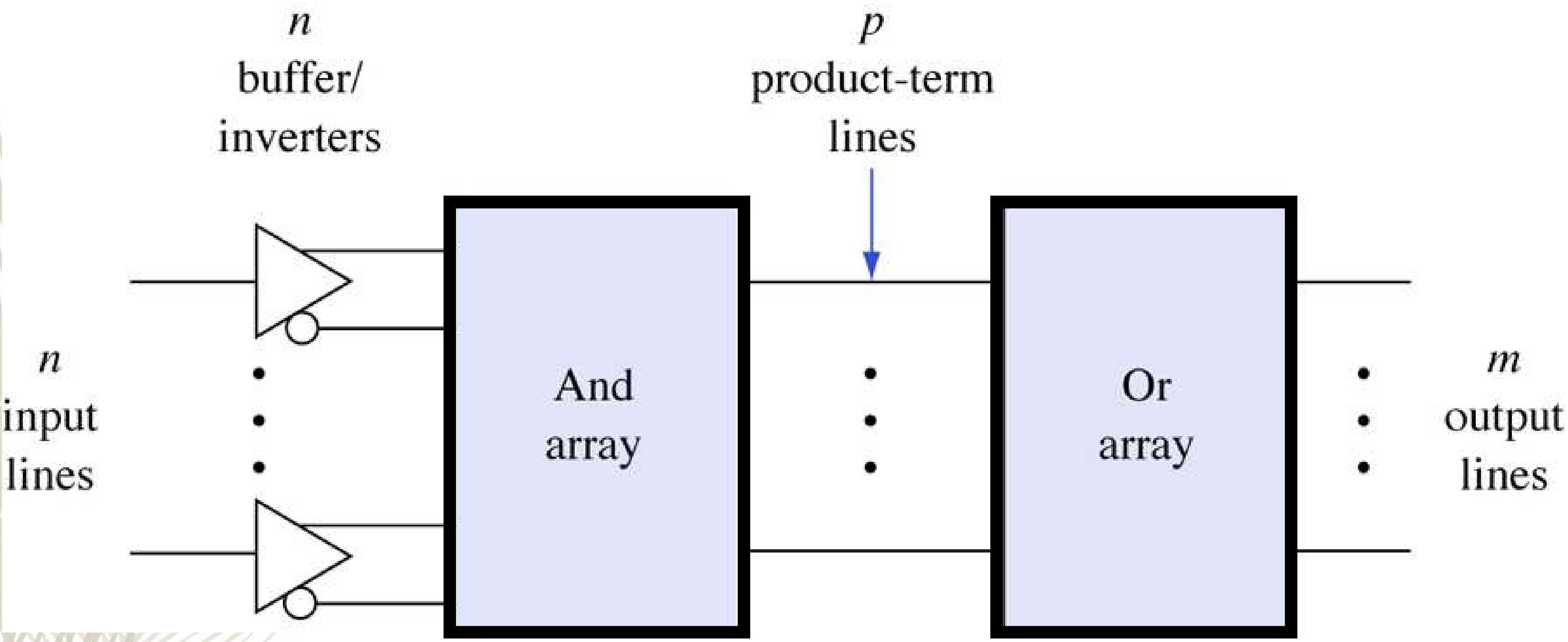
- *General purpose chip for implementing circuits*
- *Can be customized using programmable switches*
  - Main types of PLDs
    - *PLA*
    - *PAL*
    - *ROM*
    - *CPLD*
    - *FPGA*
- Custom chips: standard cells, sea of gates


PLDs
SPLDs
CPLDs
PROMs
PLAs
PALs
GALs
etc.

PLD

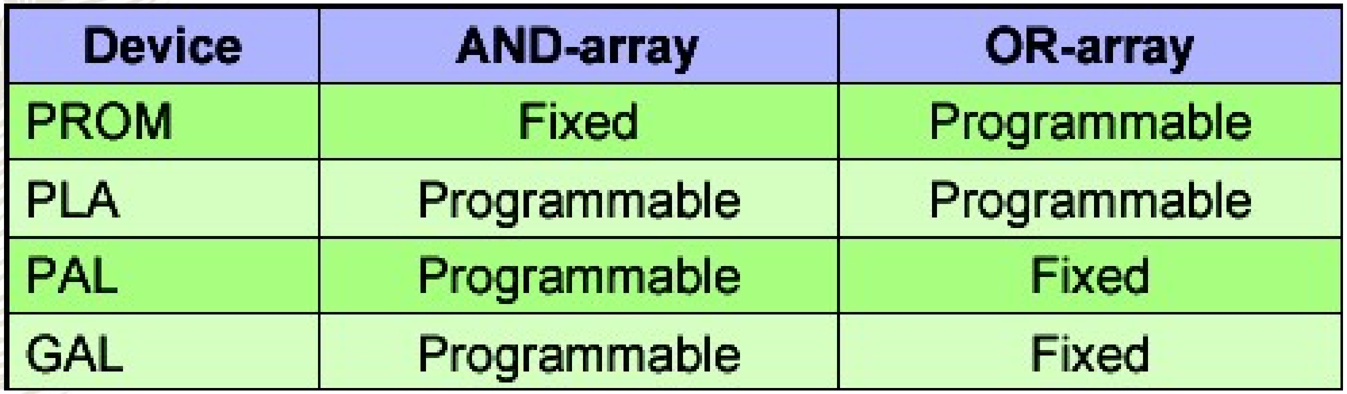
- The purpose of a PLD device is to permit elaborate digital logic designs to be implemented by the user in a single device.
- Can be erased electrically and reprogrammed with a new design, making them very well suited for academic and prototyping
- Types of Programmable Logic Devices
- SPLDs (Simple Programmable Logic Devices)
  - ROM (Read-Only Memory)
  - PLA (Programmable Logic Array)
  - PAL (Programmable Array Logic)
  - GAL (Generic Array Logic)
- CPLD (Complex Programmable Logic Device)
- FPGA (Field-Programmable Gate Array)

General structure of PLDs.



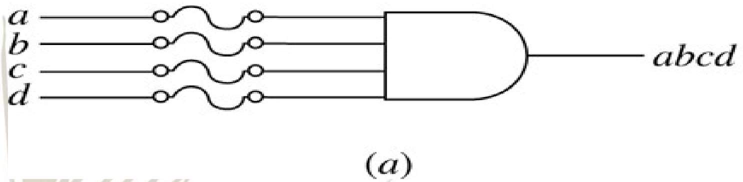
PLD

- The differences between the first three categories are these:
  - In a ROM, the input connection matrix is hardwired. The user can modify the output connection matrix.
  - In a PAL/GAL the output connection matrix is hardwired. The user can modify the input connection matrix.
  - In a PLA the user can modify both the input connection matrix and the output connection matrix.

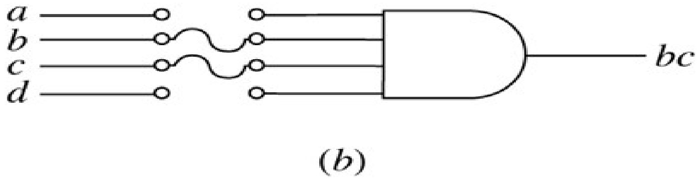


Device	AND-array	OR-array
PROM	Fixed	Programmable
PLA	Programmable	Programmable
PAL	Programmable	Fixed
GAL	Programmable	Fixed

Programming by blowing fuses.

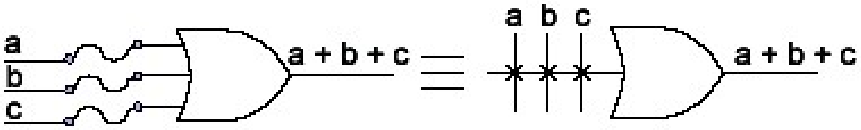


(*a*) Before programming.

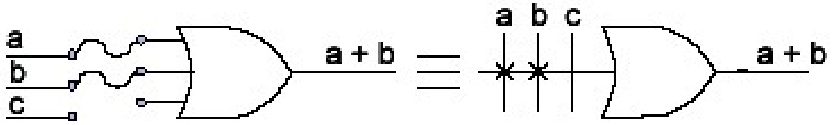


(*b*) After programming.

OR - PLD Notation

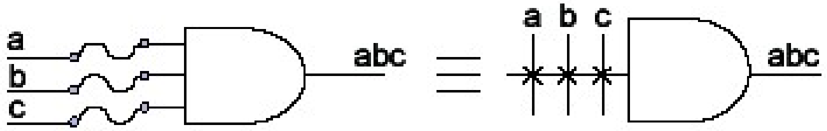


OR gate before programming

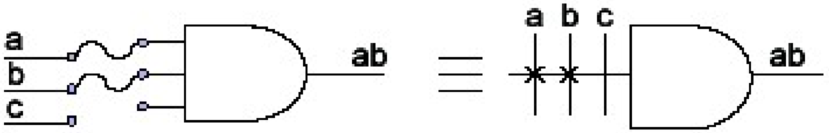


OR gate after programming

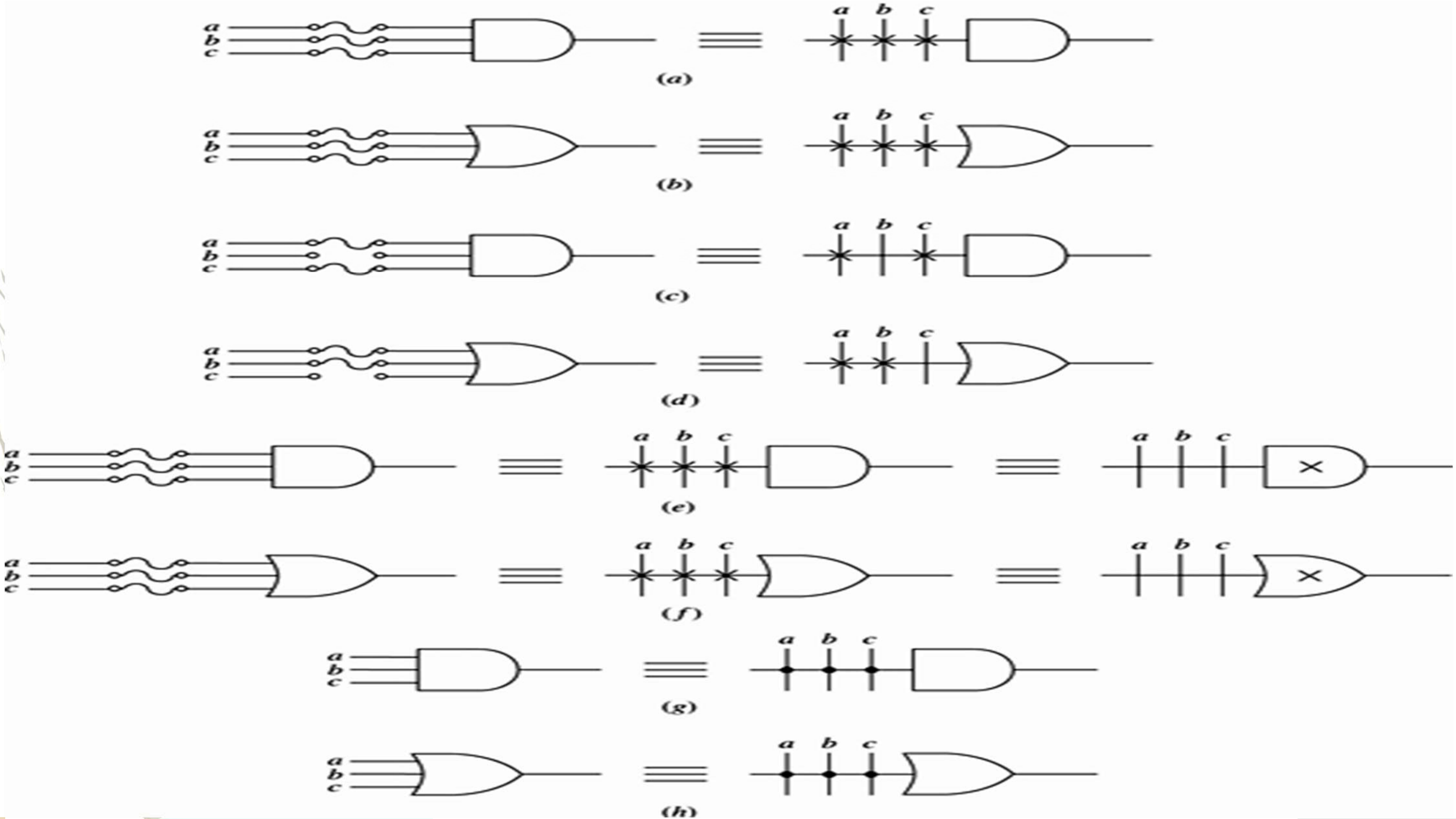
AND - PLD Notation



AND gate before programming



AND gate after programming



(a)

(b)

(c)

(d)

(e)

(f)

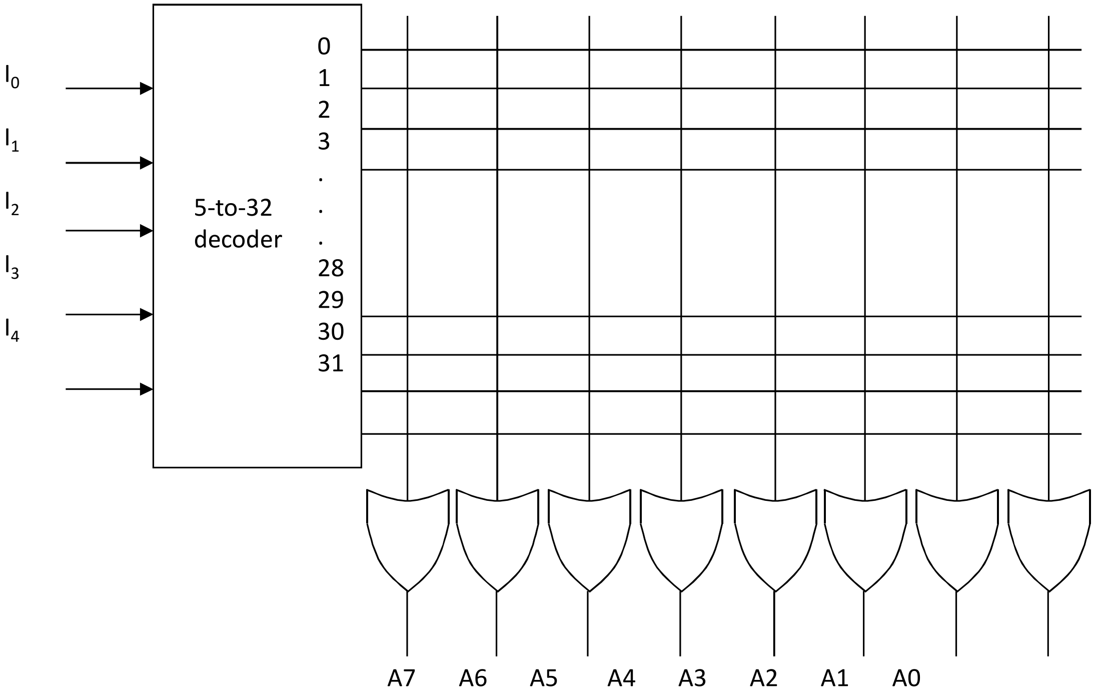
(g)

(h)

ROM Internal Logic

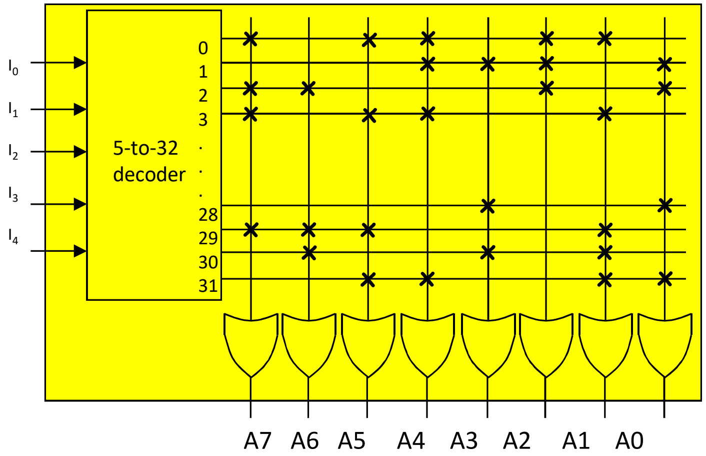
- The decoder stage produces ALL possible minterms
- 32 Words of 8 bits each
- 5 input lines (address)
- Each OR gate has a 32 input
- A contact can be made using fuse/anti-fuse

Internal Logic of a 32x8 ROM



Programming a ROM

Inputs					Outputs							
I4	**I3**	**I2**	**I1**	**I0**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		.						.				
		.						.				
		.						.				
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1



- Every ONE in truth table specifies a closed circuit
- Every ZERO in truth table specifies an OPEN circuit
- Example: At address 00011 → The word 10110010 is stored

Example 1

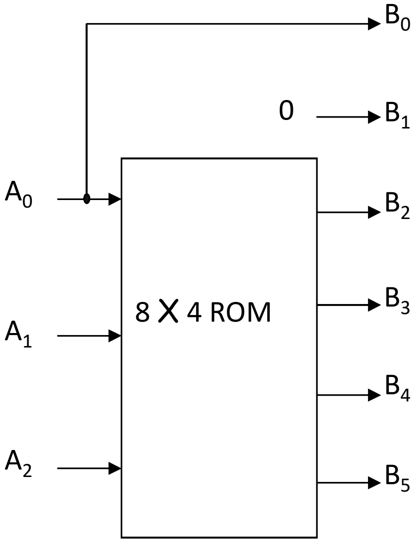
- **Example**: Design a combinational circuit using ROM. The circuit accepts a 3-bit number and generates an output binary number equal to the square of the number.
- **Solution**: Derive truth table:

Inputs			Outputs						
A2	A1	A0	B5	B4	B3	B2	B1	B0	SQ
0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1
0	1	0	0	0	0	1	0	0	4
0	1	1	0	0	1	0	0	1	9
1	0	0	0	1	0	0	0	0	16
1	0	1	0	1	1	0	0	1	25
1	1	0	1	0	0	1	0	0	36
1	1	1	1	1	0	0	0	1	49

Example 1 (cont.)

Inputs			Outputs				
A2	A1	A0	B5	B4	B3	B2	SQ
0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	1
0	1	0	0	0	0	1	4
0	1	1	0	0	1	0	9
1	0	0	0	1	0	0	16
1	0	1	0	1	1	0	25
1	1	0	1	0	0	1	36
1	1	1	1	1	0	0	49

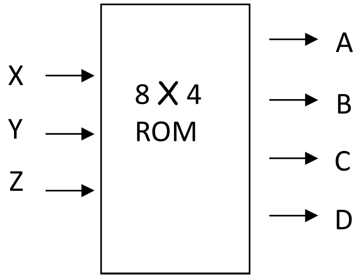
ROM truth table – specifies the required connections



- B1 is ALWAYS 0 ➔ no need to generate it using the ROM
- B0 is equal to A0 ➔ no need to generate it using the ROM
- Therefore: The minimum size of ROM needed is 2^3X4 or 8X4

Example 2

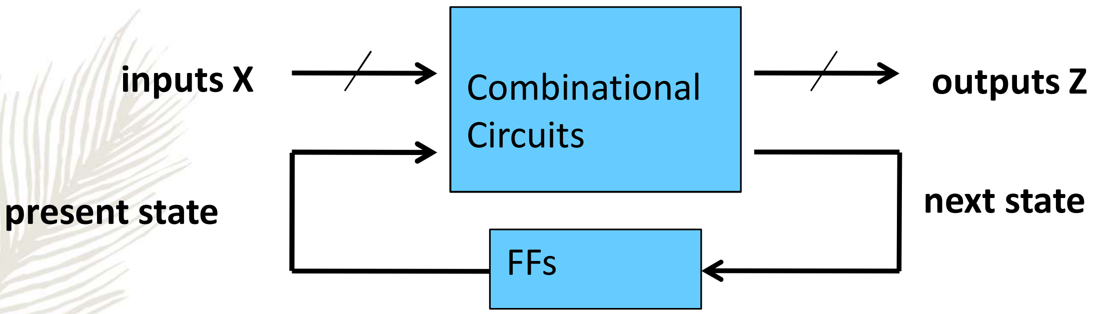
- **Problem**: Tabulate the truth for an 8 X 4 ROM that implements the following four Boolean functions:
- A(X,Y,Z) = $\Sigma m(3,6,7)$; B(X,Y,Z) = $\Sigma m(0,1,4,5,6)$
- C(X,Y,Z) = $\Sigma m(2,3,4)$; D(X,Y,Z) = $\Sigma m(2,3,4,7)$



- **Solution**:

Inputs			Outputs			
X	Y	Z	A	B	C	D
0	0	0	0	1	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	1
0	1	1	1	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	0	0
1	1	0	1	1	0	0
1	1	1	1	0	0	1

Sequential Circuit Implementation with ROM

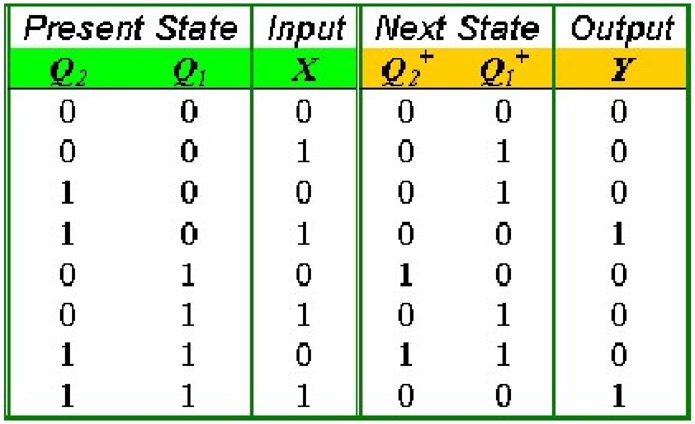


- sequential circuit = combinational circuit + memory
- Combinational part can be built with a ROM as shown previously
  - Number of address lines = No. of FF + No. of inputs
  - Number of outputs = No. of FF + No. of outputs

Example

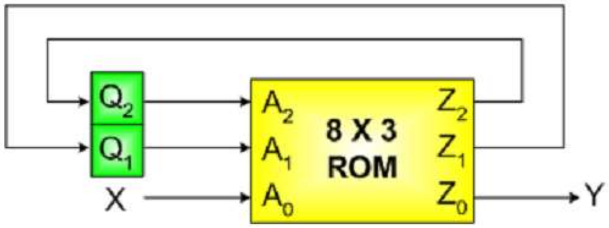
- **Example:** Design a sequential circuit whose state table is given, using a ROM and a register.

- **State Table**



Present State		Input	Next State		Output
Q_2	Q_1	X	Q_2^+	Q_1^+	Y
0	0	0	0	0	0
0	0	1	0	1	0
1	0	0	0	1	0
1	0	1	0	0	1
0	1	0	1	0	0
0	1	1	0	1	0
1	1	0	1	1	0
1	1	1	0	0	1

We need a 8x3 ROM (why?)
3 address lines and 3 data lines

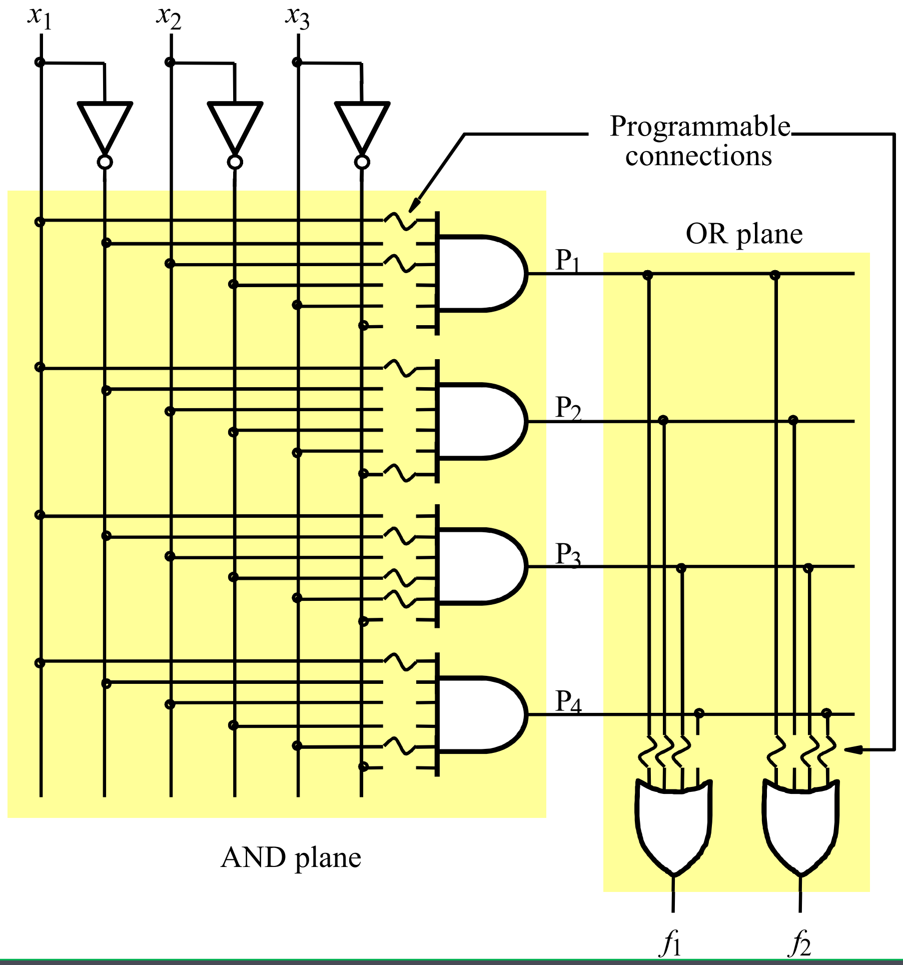


Exercise: Compare design with ROMs with the traditional design procedure.

Gate Level Version of PLA

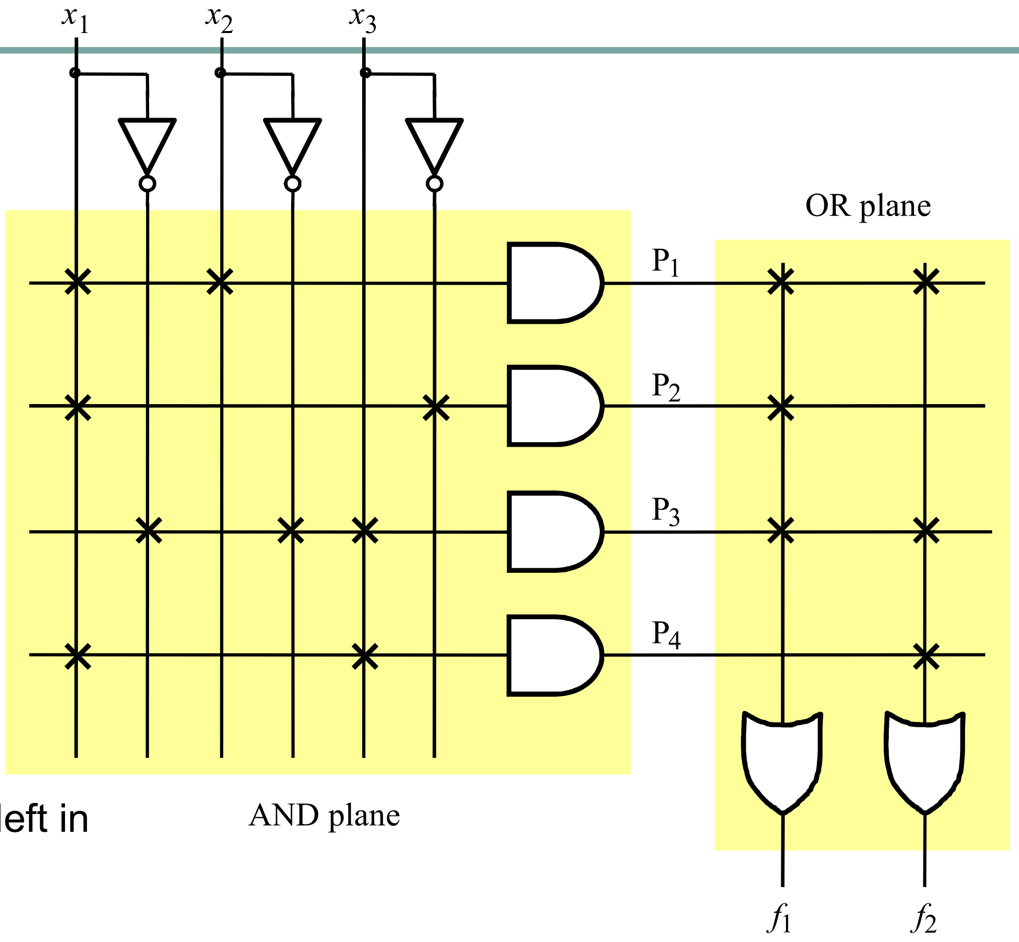
$f_1 = x_1x_2+x_1x_3'+x_1'x_2'x_3$

$f_2 = x_1x_2+x_1'x_2'x_3+x_1x_3$



$f_1 = x_1x_2+x_1x_3'+x_1'x_2'x_3$

$f_2 = x_1x_2+x_1'x_2'x_3+x_1x_3$



x marks the connections left in place after programming

Limitations of PLAs

- PLAs come in various sizes
  - *Typical size is 16 inputs, 32 product terms, 8 outputs*
    - Each AND gate has large fan-in → this limits the number of inputs that can be provided in a PLA
    - 16 inputs → 3^{16} = possible input combinations; only 32 permitted (since 32 AND gates) in a typical PLA
    - 32 AND terms permitted → large fan-in for OR gates as well
      - *This makes PLAs slower and slightly more expensive than some alternatives to be discussed shortly*
    - 8 outputs → could have shared minterms, but not required

Design for PLA:Example

– Implement the following functions using PLA

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

Personality Matrix

Product term	Inputs			Outputs			
	A	B	C	F0	F1	F2	F3
~~A~~ B	1	1	-	0	1	1	0
B ~~C~~	-	0	1	0	0	0	1
~~A C~~	1	-	0	0	1	0	0
B C	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

Reuse of terms

Input Side:

1 = asserted in term
0 = negated in term
- = does not participate

Output Side:

1 = term connected to output
0 = no connection to output

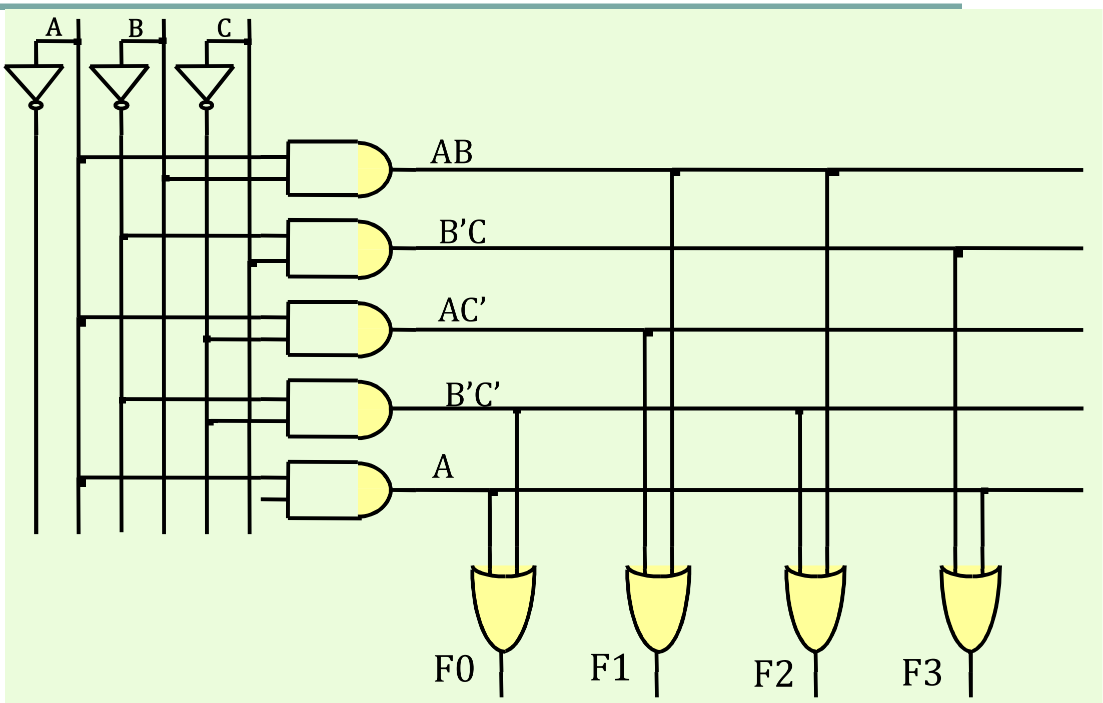
Example: Continued

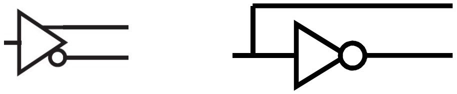
$$F0 = A + B'C'$$
$$F1 = AC' + AB$$
$$F2 = B'C' + AB$$
$$F3 = B'C + A$$

Personality Matrix

Product term	Inputs			Outputs			
	A	B	C	F_0	F_1	F_2	F_3
AB	1	1	-	0	1	1	0
$\overline{B}C$	-	0	1	0	0	0	1
$A\overline{C}$	1	-	0	0	1	0	0
$\overline{B}\,\overline{C}$	-	0	0	1	0	1	0
A	1	-	-	1	0	0	1

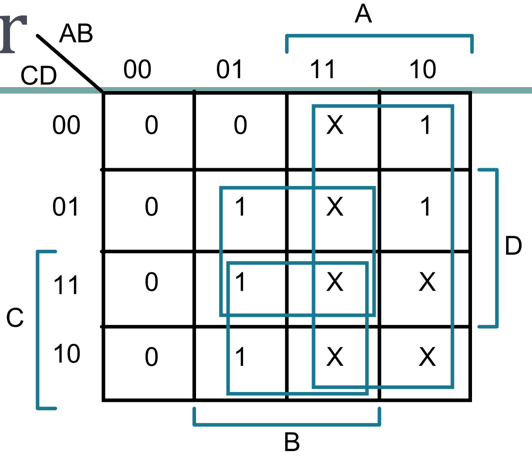
Reuse of terms





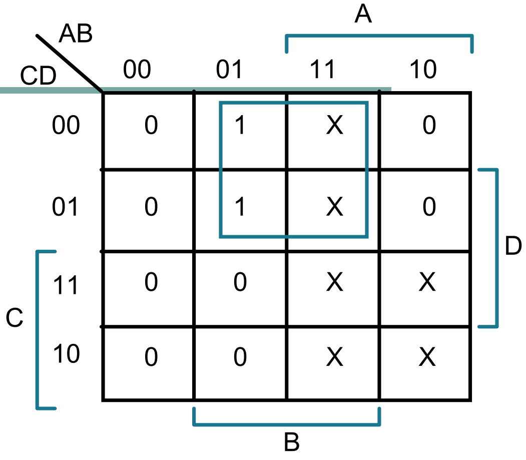
BCD to Gray Code Converter

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	1	1	1	0
0	1	1	0	1	0	1	0
0	1	1	1	1	0	1	1
1	0	0	0	1	0	0	1
1	0	0	1	1	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X



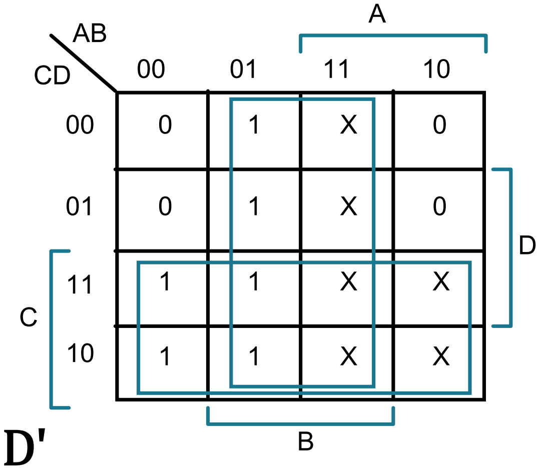
CD \ AB	00	01	11	10
00	0	0	X	1
01	0	1	X	1
11	0	1	X	X
10	0	1	X	X

K-map for W



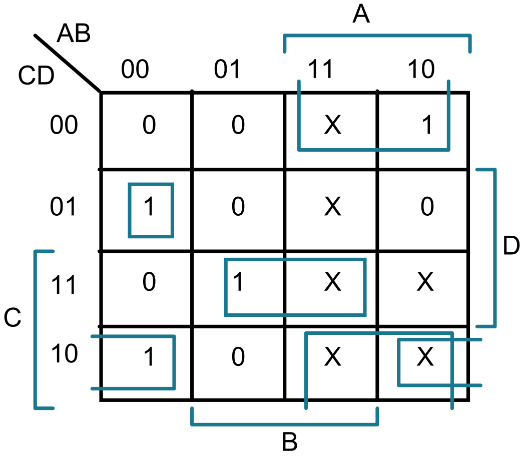
CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	0	0	X	X
10	0	0	X	X

K-map for X



CD \ AB	00	01	11	10
00	0	1	X	0
01	0	1	X	0
11	1	1	X	X
10	1	1	X	X

K-map for Y



CD \ AB	00	01	11	10
00	0	0	X	1
01	1	0	X	0
11	0	1	X	X
10	1	0	X	X

K-map for Z

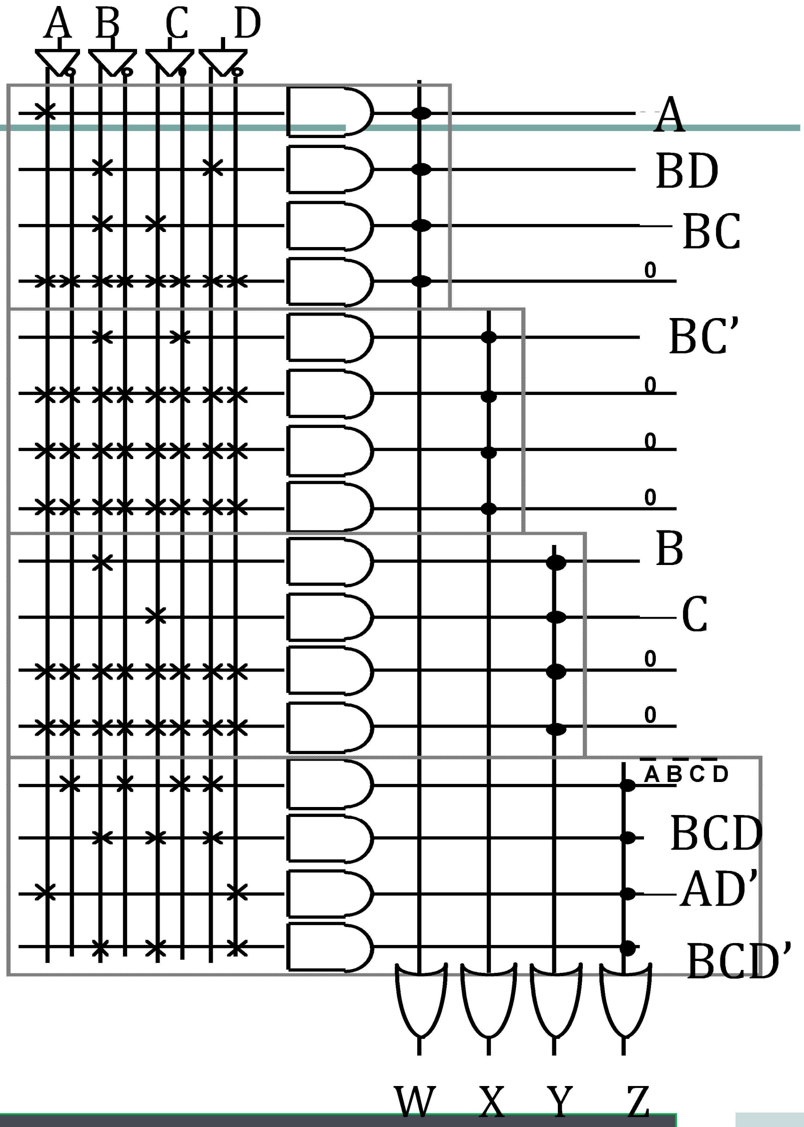
Minimized Functions:

$$W = A + B\,D + B\,C$$
$$X = B\,C'$$
$$Y = B + C$$
$$Z = A'B'C'D + B\,C\,D + A\,D' + B'\,C\,D'$$

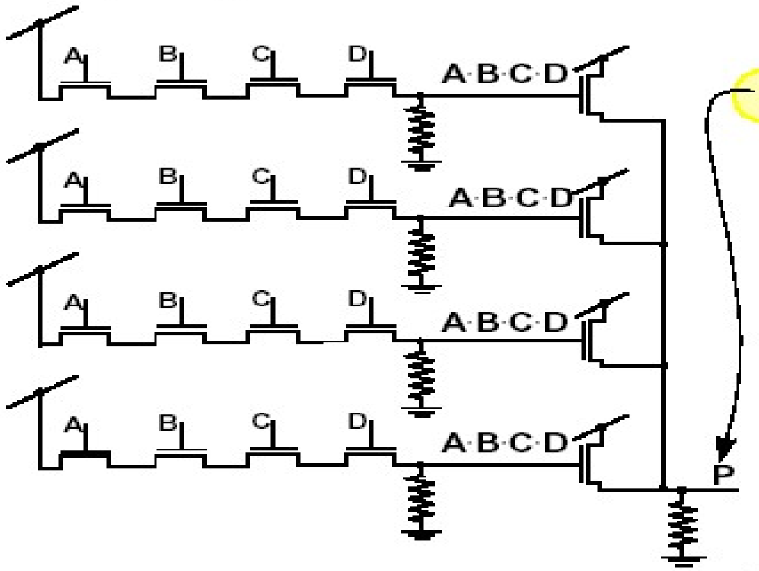
4 product terms per each OR gate

Product terms cannot be shared !

PLA achieves higher flexibility at the cost of lower speed!



Array Logic

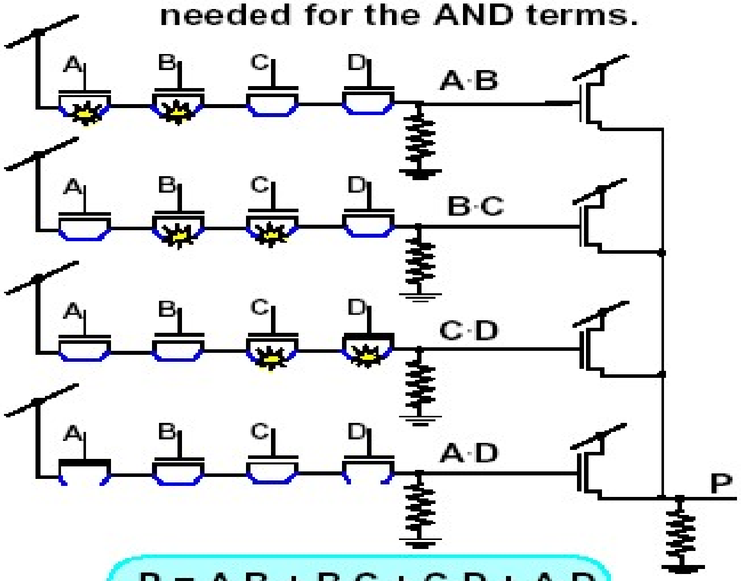


Combine the AND and OR logic

$P = A·B·C·D + A·B·C·D + A·B·C·D + A·B·C·D$

A Fuse Programmed Logic Array

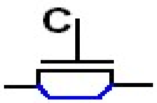
Program the logic
Short all transistors with fuses.
Remove shorts from transistors needed for the AND terms.



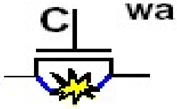
$P = A·B + B·C + C·D + A·D$

Fuse Programming

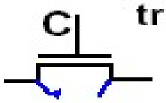
Start with all transistors shorted.



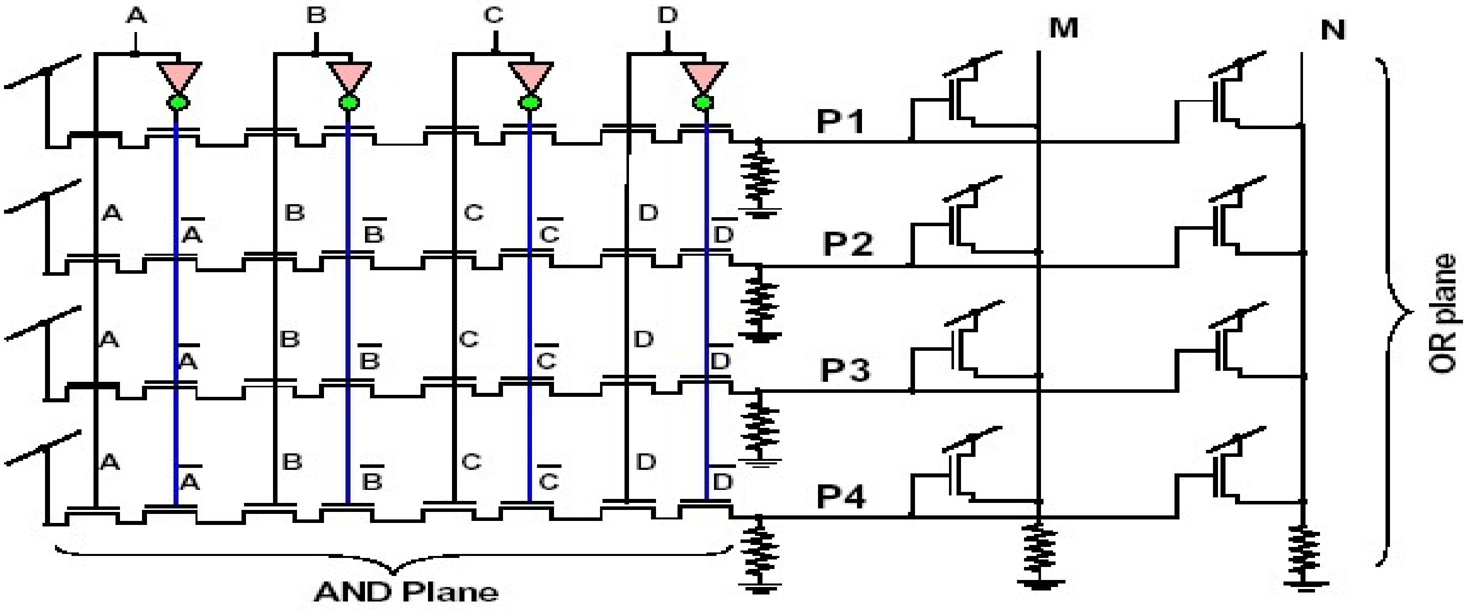
Zap the short where you want a transistor



Leave a working transistor

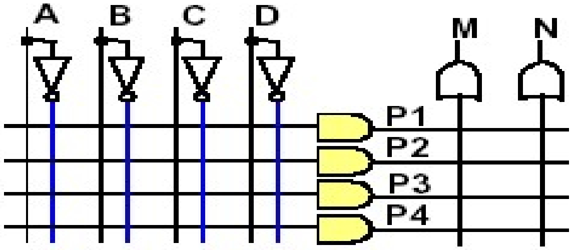


Programmable Logic Array (PLA)



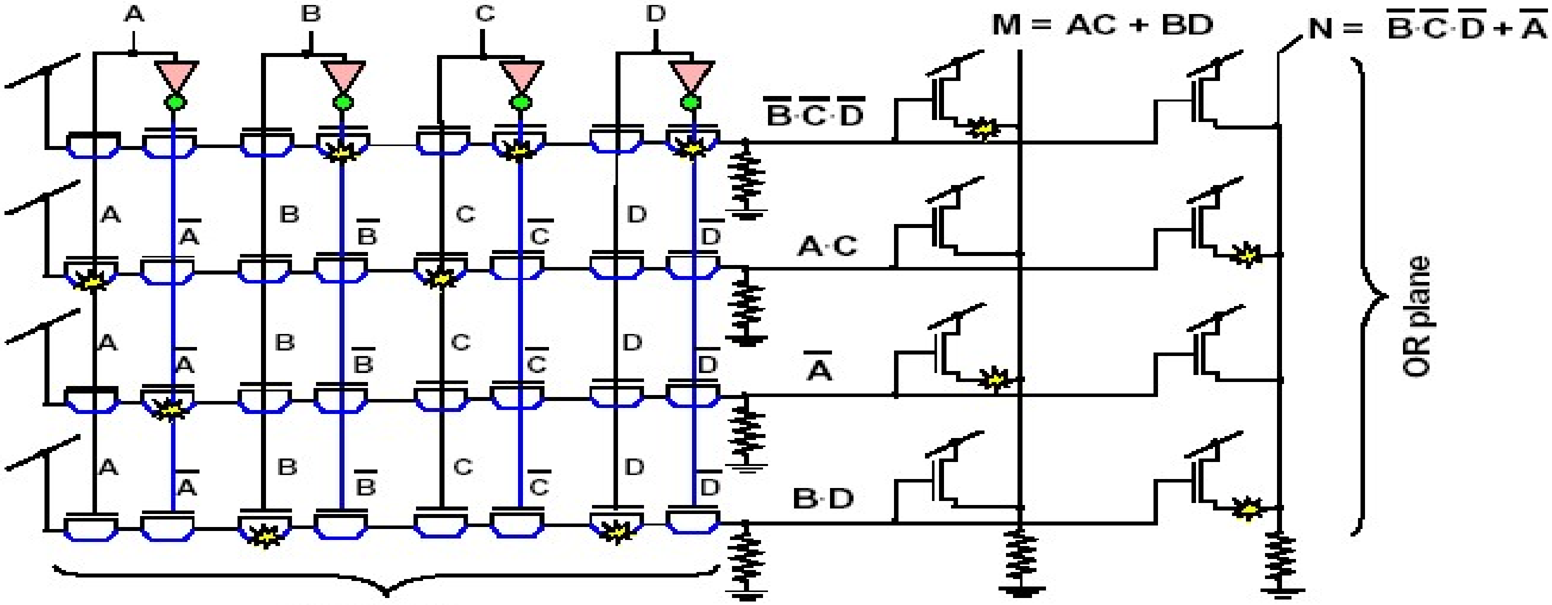
Two outputs M and N
Can implement two Σ of Π functions
with 4 inputs which includes x and $\bar{x}$
and with 4 product terms, P1, P2, P3 and P4.

Use the simple symbol for logic design



Symbolic PLA

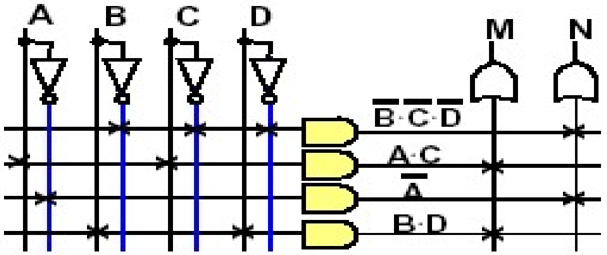
Programming the PLA


A
B
C
D
M = AC + BD
N = $\overline{B}\cdot\overline{C}\cdot\overline{D}+\overline{A}$
$\overline{B}\cdot\overline{C}\cdot\overline{D}$
A·C
$\overline{A}$
B·D
A
$\overline{A}$
B
$\overline{B}$
C
$\overline{C}$
D
$\overline{D}$
OR plane
AND Plane

AND Plane
Blow fuses for letters you want in the AND term

OR Plane
Blow fuses to remove unwanted ORs

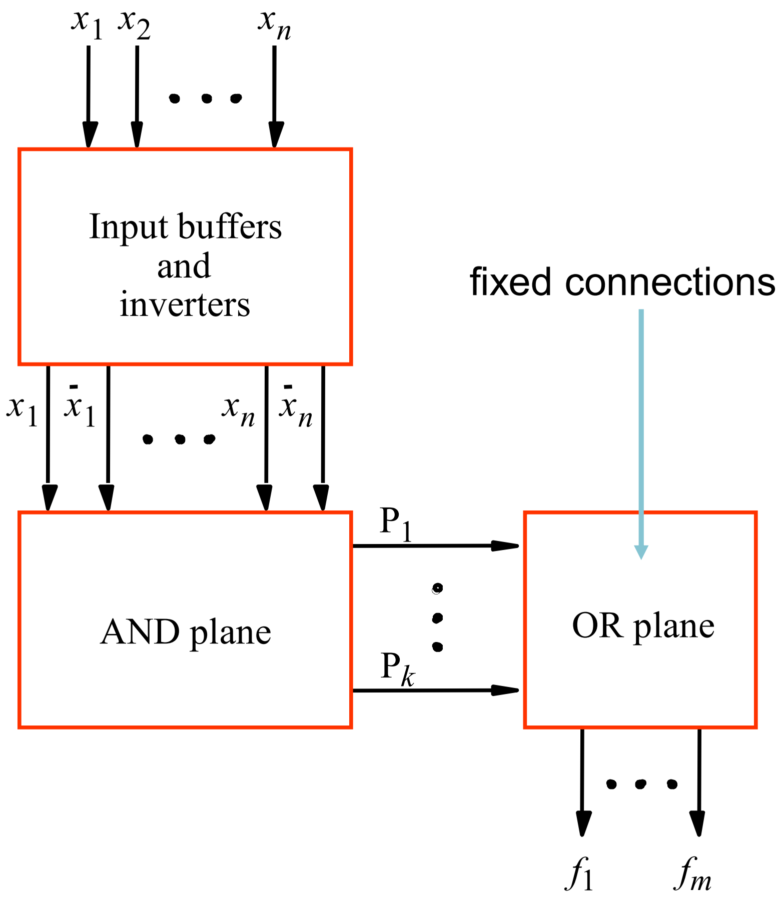
Symbolic PLA
Put an × where you want a connection



Symbolic PLA

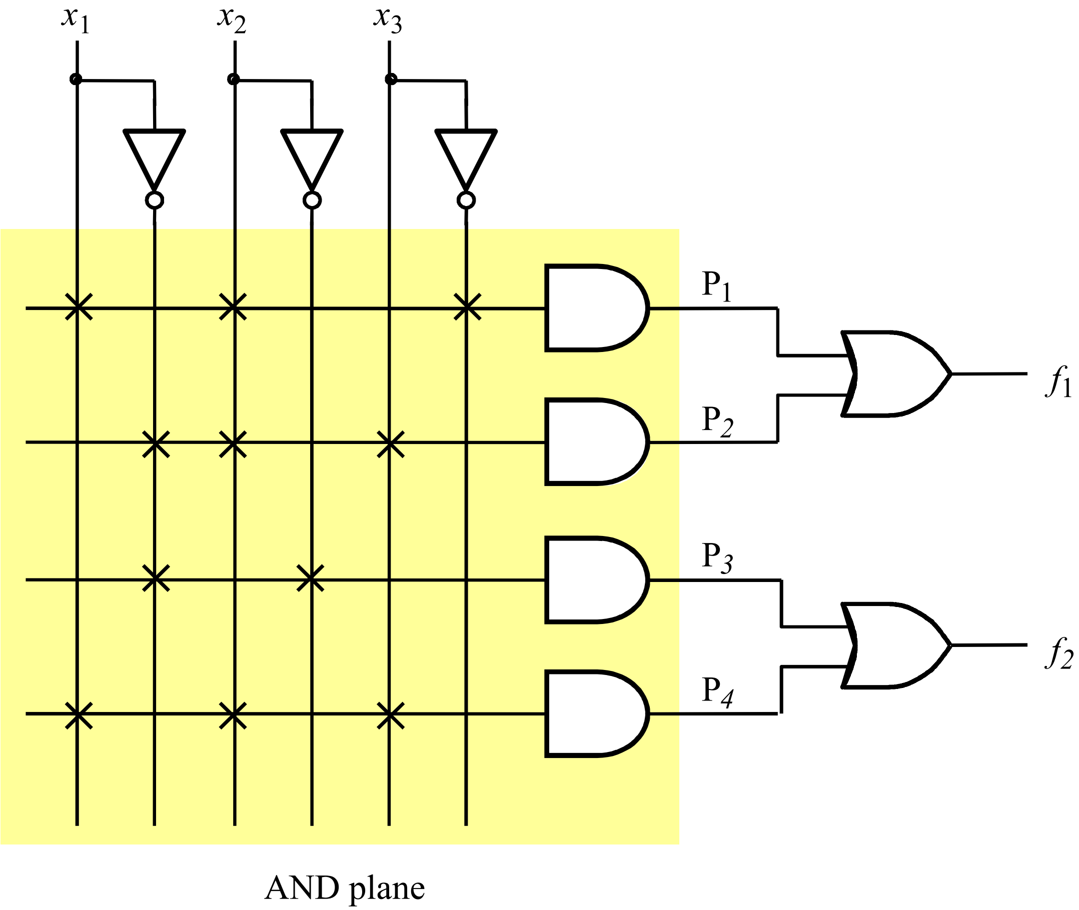
Programmable Array Logic (PAL)

- Also used to implement circuits in SOP form
- The connections in the AND plane are programmable
- The connections in the OR plane are <u>NOT</u> programmable

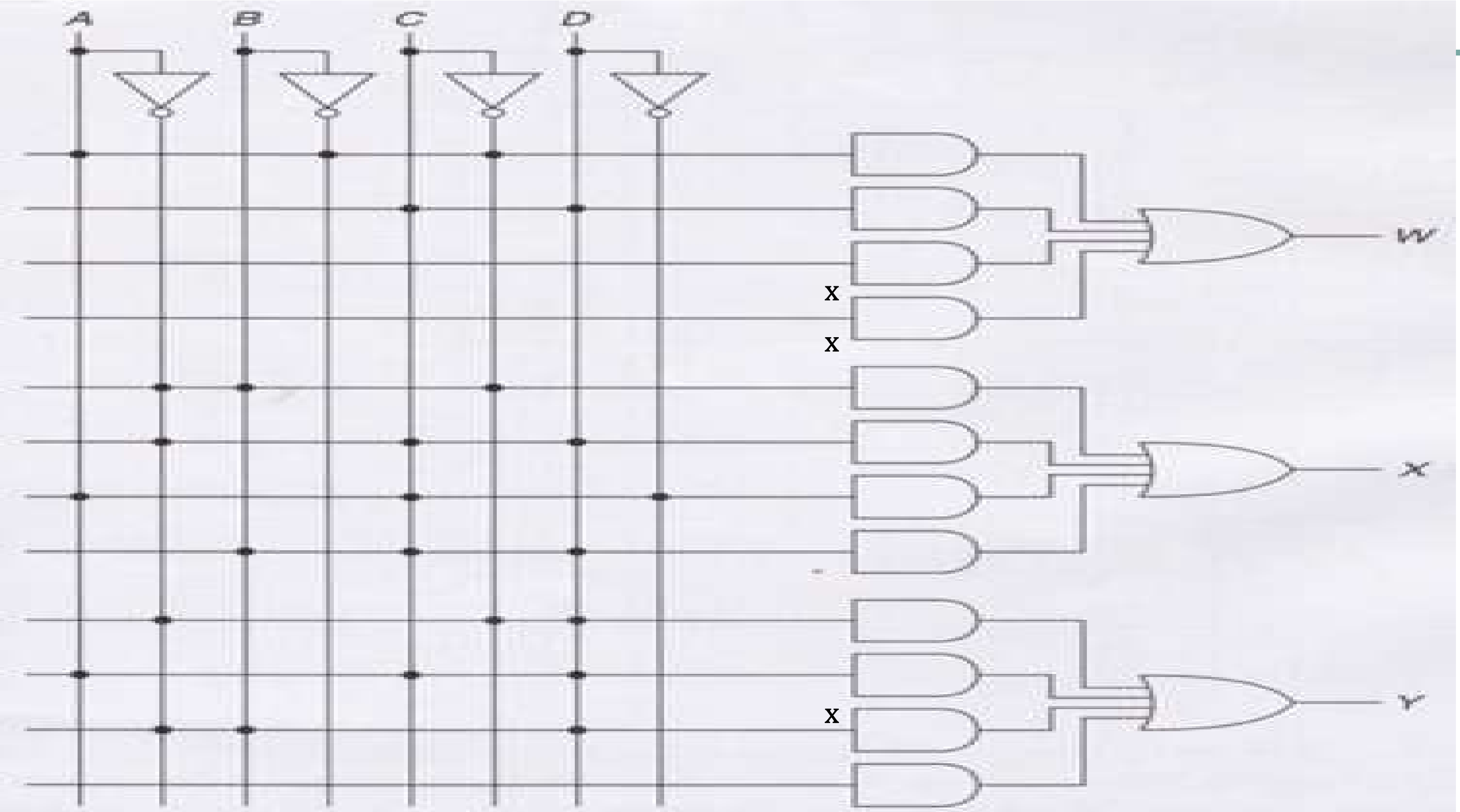


$f_1 = x_1x_2x_3'+x_1'x_2x_3$

$f_2 = x_1'x_2'+x_1x_2x_3$



PAL



$$W = AB'C \quad + CD$$
$$X = A'BC \quad + A'CD + ACD' \ + BCD$$
$$Y = A'CD + ACD + A'BD$$

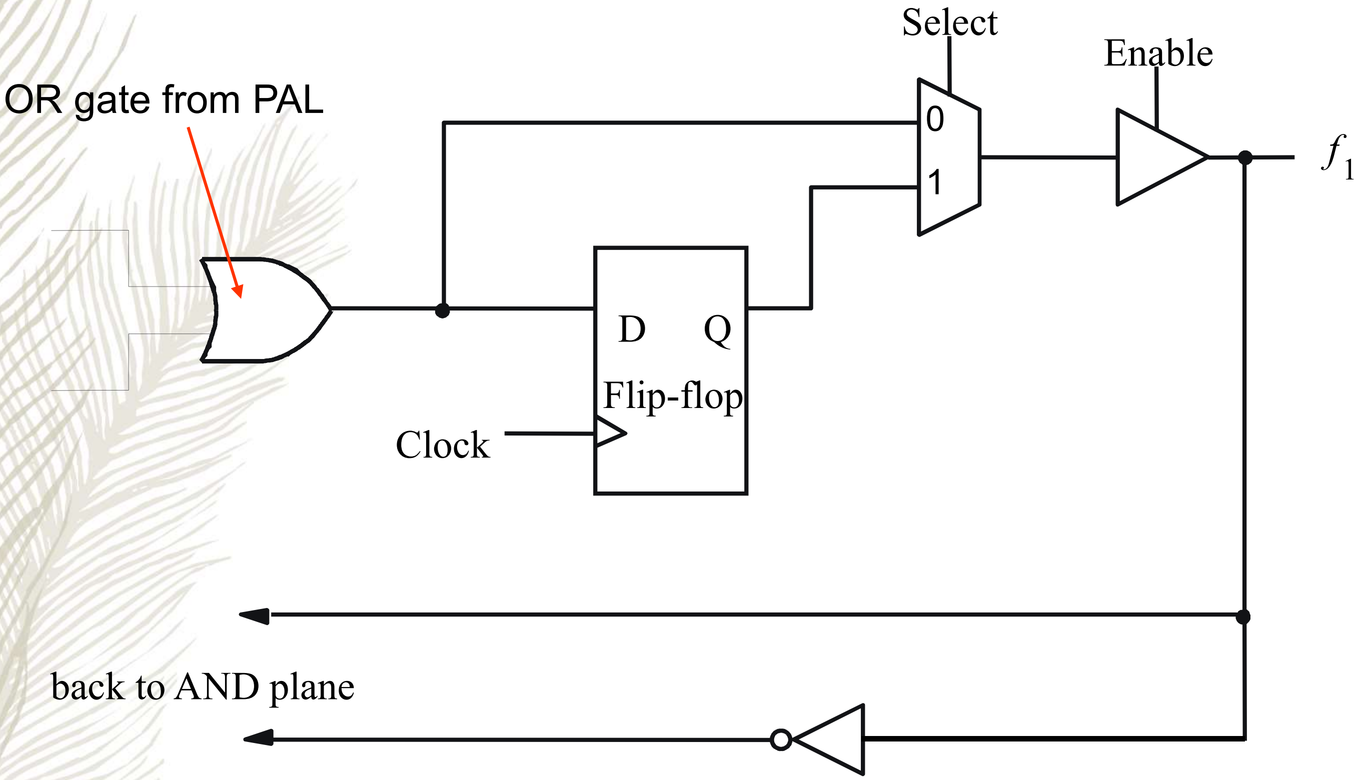
Comparing PALs and PLAs

- PALs have the same limitations as PLAs (small number of allowed AND terms) plus they have a fixed OR plane → less flexibility than PLAs

- PALs are simpler to manufacture, cheaper, and faster (better performance)

- PALs also often have extra circuitry connected to the output of each OR gate
  - *The OR gate plus this circuitry is called a macrocell*

Macro cell

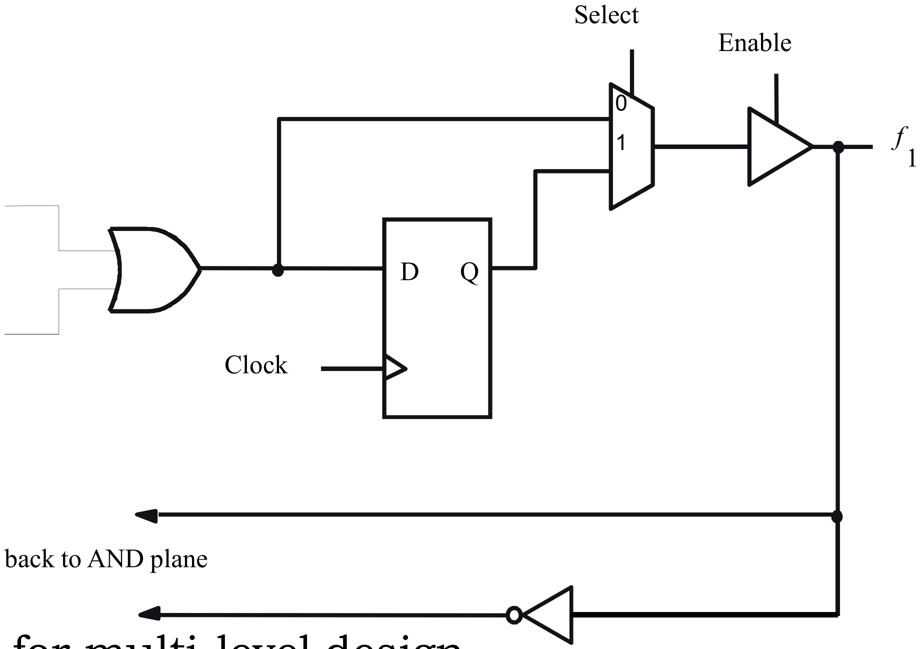

Select
Enable
OR gate from PAL
0
1
f_1
D
Q
Flip-flop
Clock
back to AND plane

Macrocell Functions

- Enable = 0 can be used to allow the output pin for f_1 to be used as an additional input pin to the PAL

- Enable = 1, Select = 0 is normal for typical PAL operation

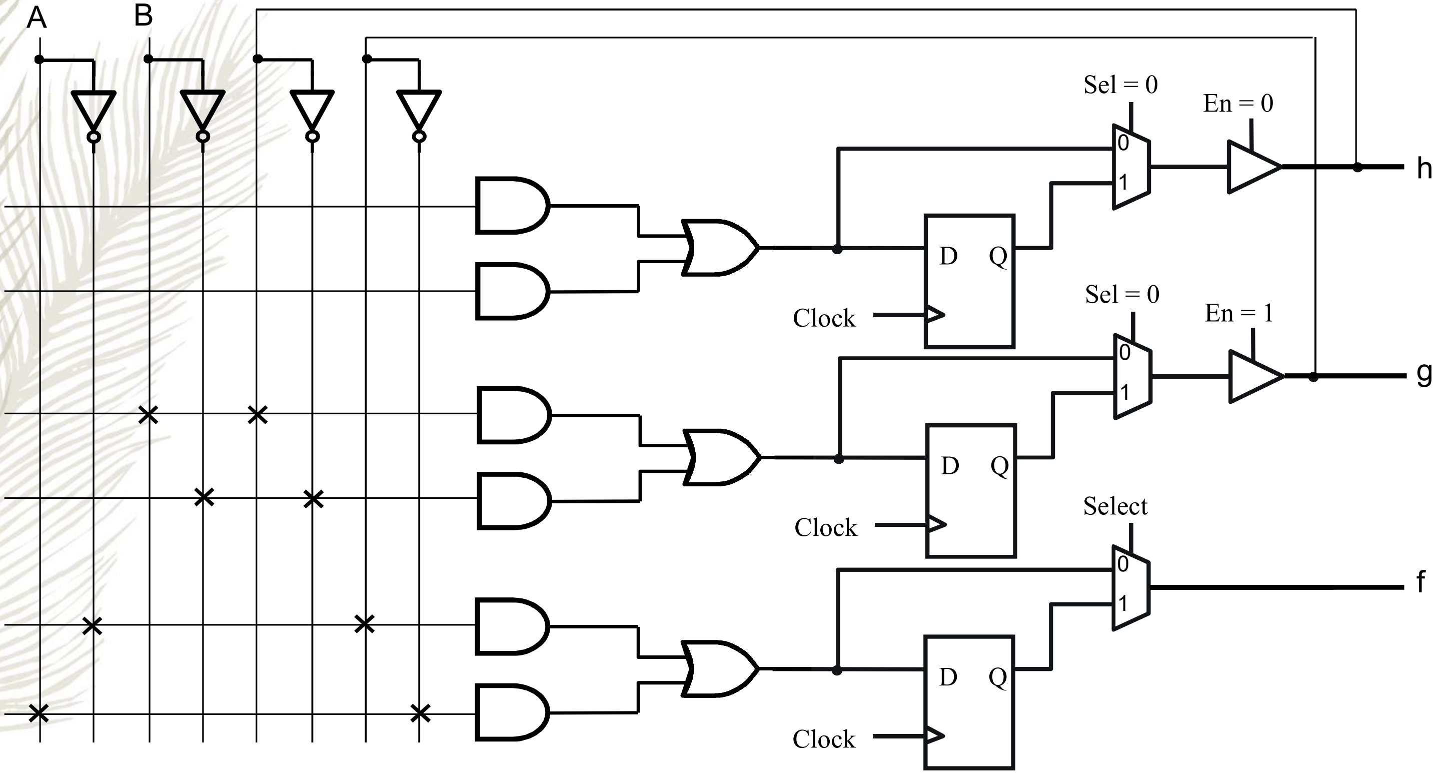
- Enable = Select = 1 allows the PAL to synchronize the output changes with a clock pulse



- The feedback to the AND plane provides for multi-level design

Multi-Level Design with PALs

- f = A'BC + A'B'C' + ABC' + AB'C = A'g + Ag'
  - *where g = BC + B'C' and C = h below*



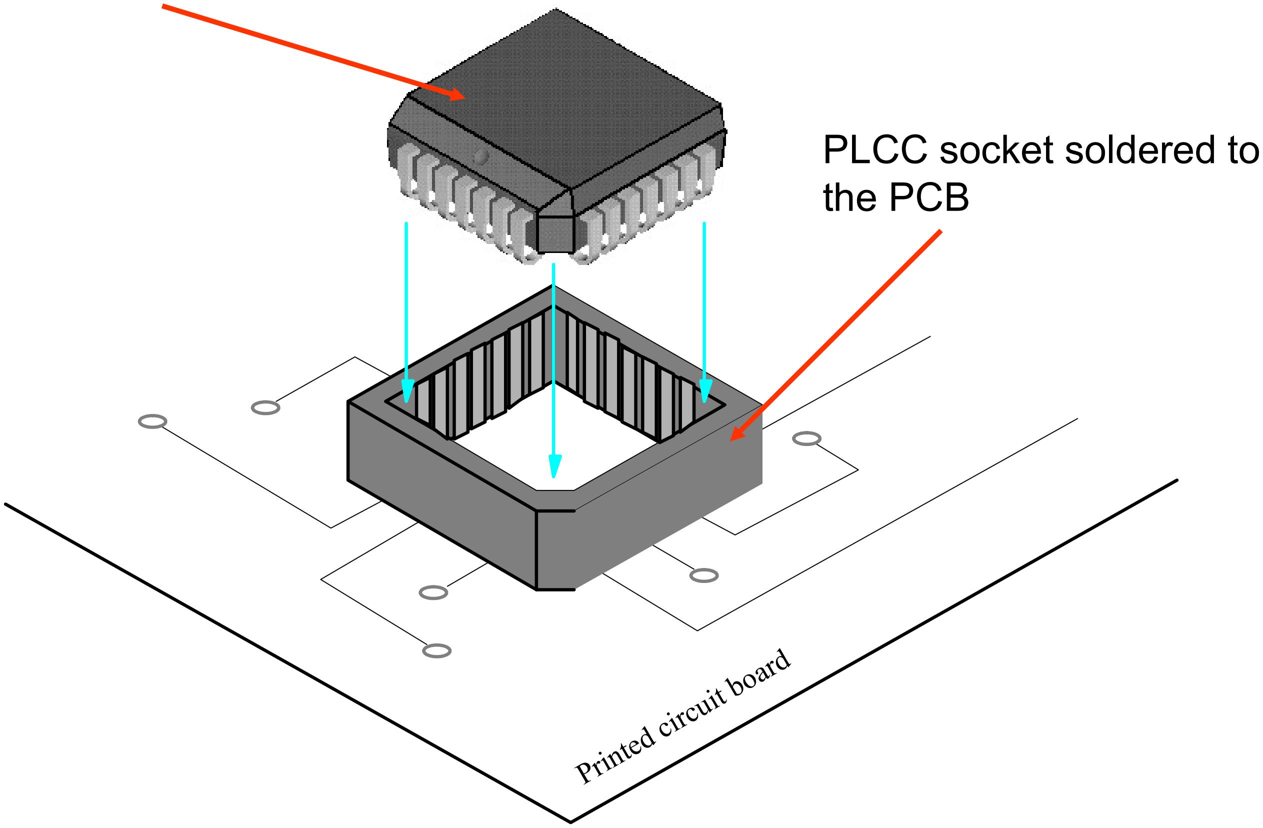
Programming SPLDs

- PLAs, PALs, and ROMs are also called SPLDs – Simple Programmable Logic Devices
- SPLDs must be programmed so that the switches are in the correct places
  - *CAD tools are usually used to do this*
    - A fuse map is created by the CAD tool and then that map is downloaded to the device via a special programming unit
  - *There are two basic types of programming techniques*
    - Removable sockets on a PCB
    - In system programming (ISP) on a PCB
      - *This approach is not very common for PLAs and PALs but it is quite common for more complex PLDs*

- An SPLD Programming Unit

- The SPLD is removed from the PCB, placed into the unit and programmed there

- Removable SPLD Socket Package
  - PLCC (plastic-leaded chip carrier)



PLCC socket soldered to the PCB

In System Programming (ISP)

- Used when the SPLD cannot be removed from the PCB

- A special cable and PCB connection are required to program the SPLD from an attached computer

- Very common approach to programming more complex PLDs like CPLDs, FPGAs, etc.

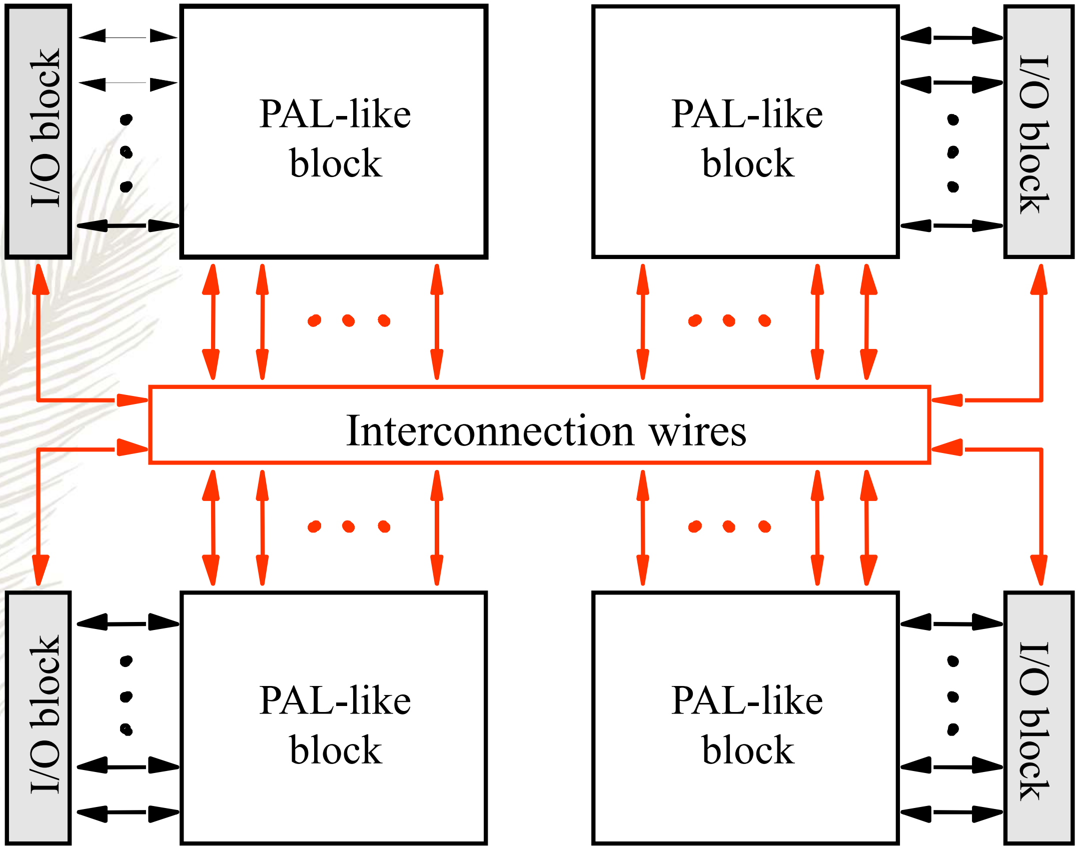
FPGA AND CPLD

- FPGA - Field-Programmable Gate Array.
- CPLD - Complex Programmable Logic Device
- FPGA and CPLD is an advance PLD.
- Support thousands of gate where as PLD only support hundreds of gates.

CPLD

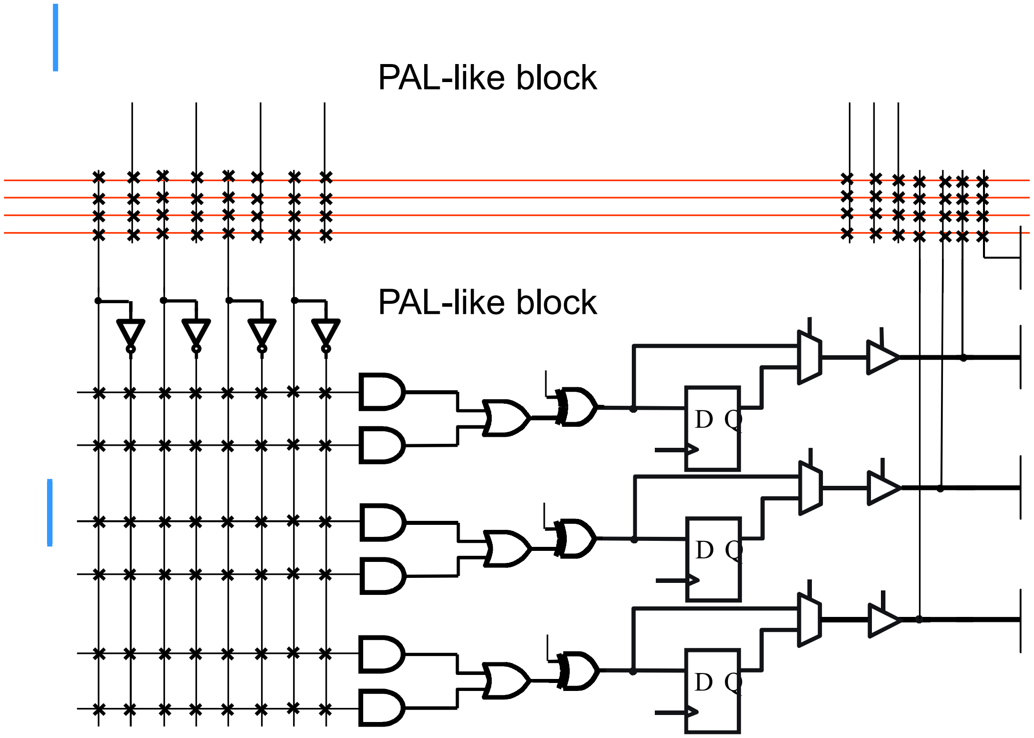
- Complex Programmable Logic Devices (CPLD)
- SPLDs (PLA, PAL) are limited in size due to the small number of input and output pins and the limited number of product terms
  - *Combined number of inputs + outputs < 32 or so*
- CPLDs contain multiple circuit blocks on a single chip
  - *Each block is like a PAL: PAL-like block*
  - *Connections are provided between PAL-like blocks via an interconnection network that is programmable*
  - *Each block is connected to an I/O block as well*

Structure of a CPLD



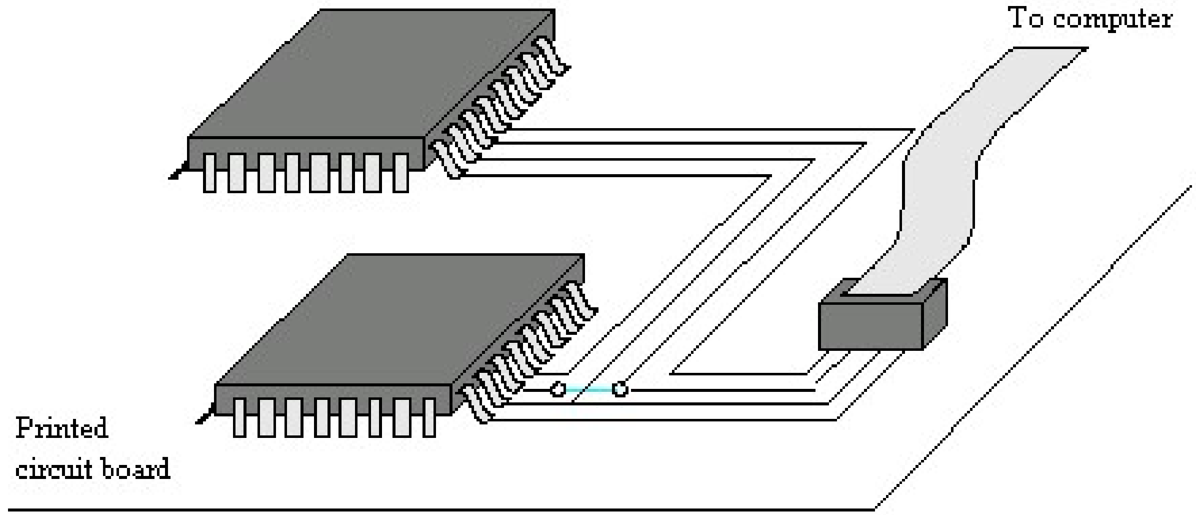
Internal Structure of a PAL-like Block

- Includes macrocells
  - *Usually about 16 each*
- Fixed OR planes
  - *OR gates have fan-in between 5-20*
- XOR gates provide negation ability
  - *XOR has a control input*



Programming a CPLD

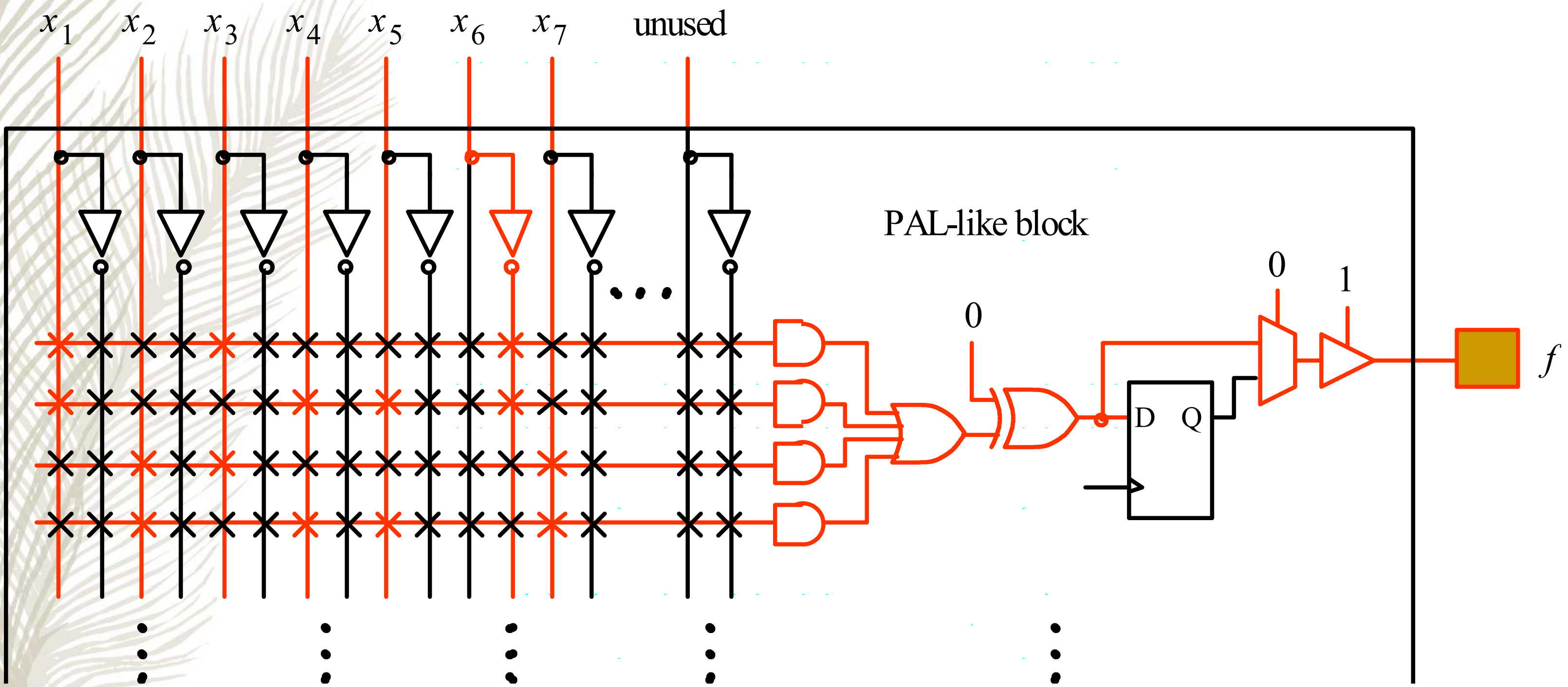
- CPLDs have many pins – large ones have > 200
  - *Removal of CPLD from a PCB is difficult without breaking the pins*
  - *Use ISP (in system programming) to program the CPLD*
  - *JTAG (Joint Test Action Group) port used to connect the CPLD to a computer*



Example CPLD

- Use a CPLD to implement the function
  - $f = x_1x_3x_6' + x_1x_4x_5x_6' + x_2x_3x_7 + x_2x_4x_5x_7$

(from interconnection wires)



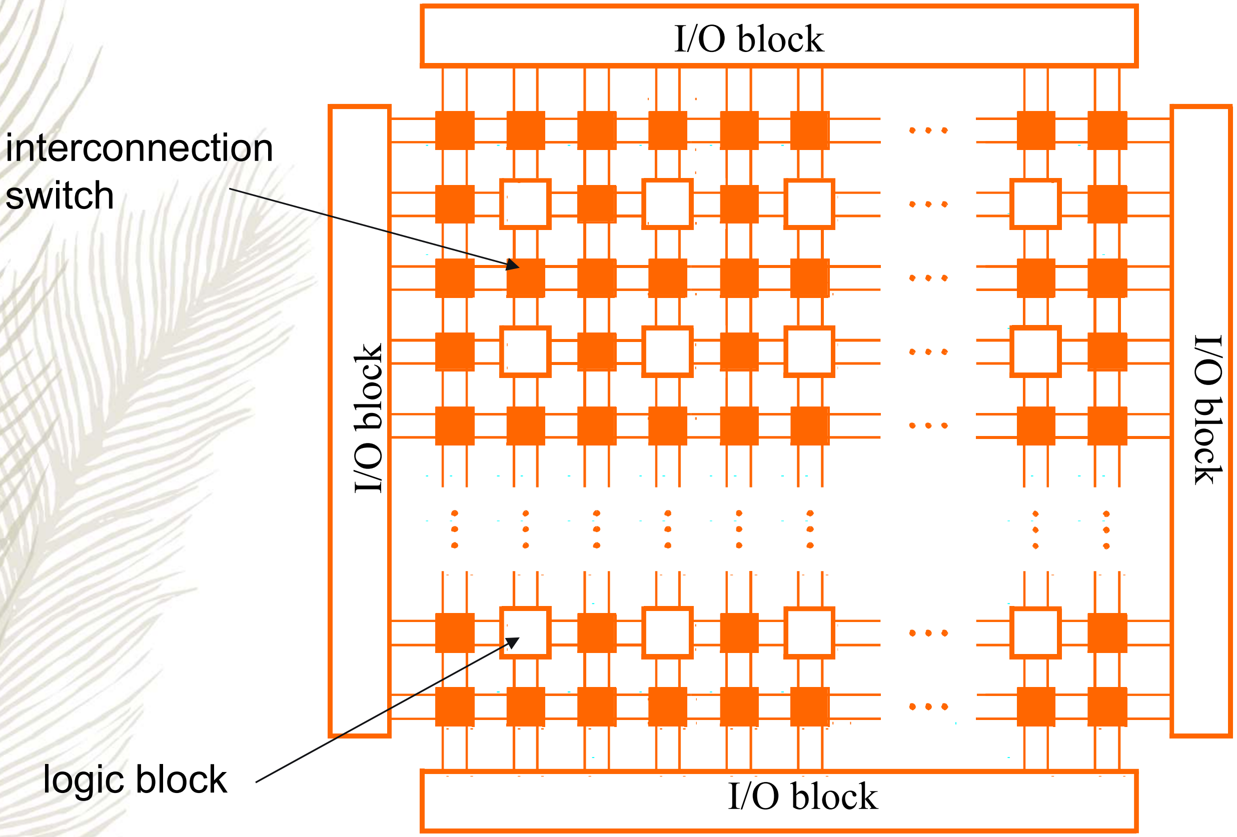
FPGA

- SPLDs and CPLDs are relatively small and useful for simple logic devices
  - *Up to about 20000 gates*

- Field Programmable Gate Arrays (FPGA) can handle larger circuits
  - *No AND/OR planes*
  - *Provide logic blocks, I/O blocks, and interconnection wires and switches*

  - *Logic blocks provide functionality*
  - *Interconnection switches allow logic blocks to be connected to each other and to the I/O pins*

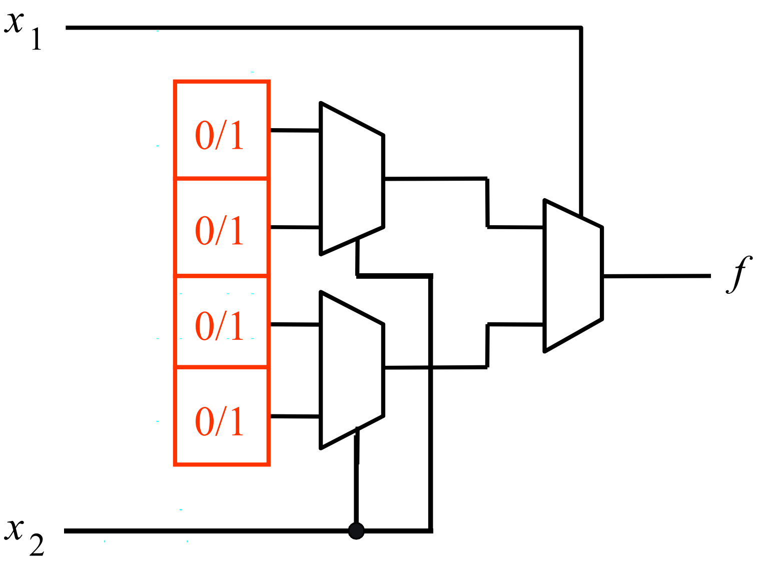
Structure of an FPGA



LUTs

- Logic blocks are implemented using a lookup table (LUT)
  - *Small number of inputs, one output*
  - *Contains storage cells that can be loaded with the desired values*

- *A 2 input LUT uses 3 MUXes to implement any desired function of 2 variables*
  - Shannon's expansion at work!

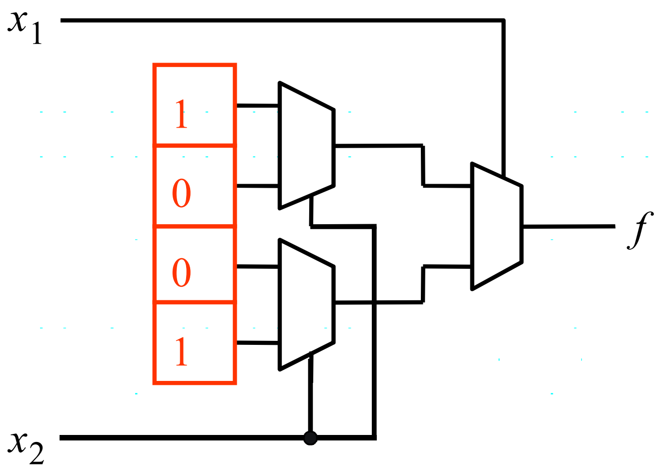


Example 2 Input LUT

x_1	x_2	f
0	0	1
0	1	0
1	0	0
1	1	1

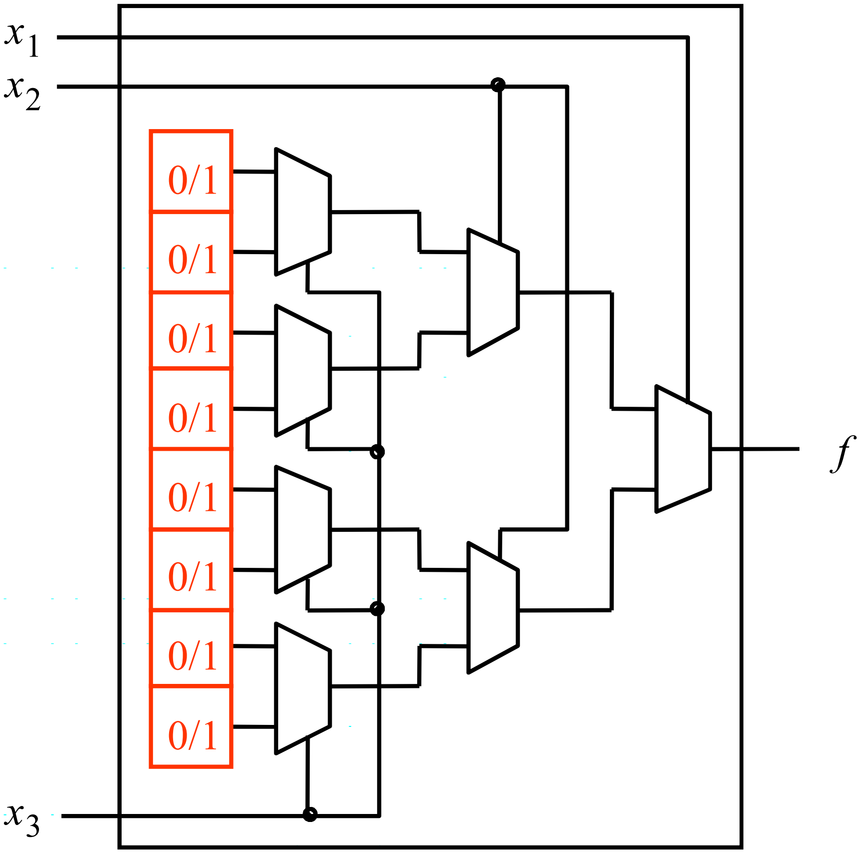
f = $x_1'x_2' + x_1x_2$, or using Shannon's expansion:

$$f = x_1'(x_2') + x_1(x_2)$$
$$= x_1'(x_2'(1) + x_2(0)) + x_1(x_2'(0) + x_2(1))$$



3 Input LUT

- 7 2x1 MUXes and 8 storage cells are required
- Commercial LUTs have 4-5 inputs, and 16-32 storage cells

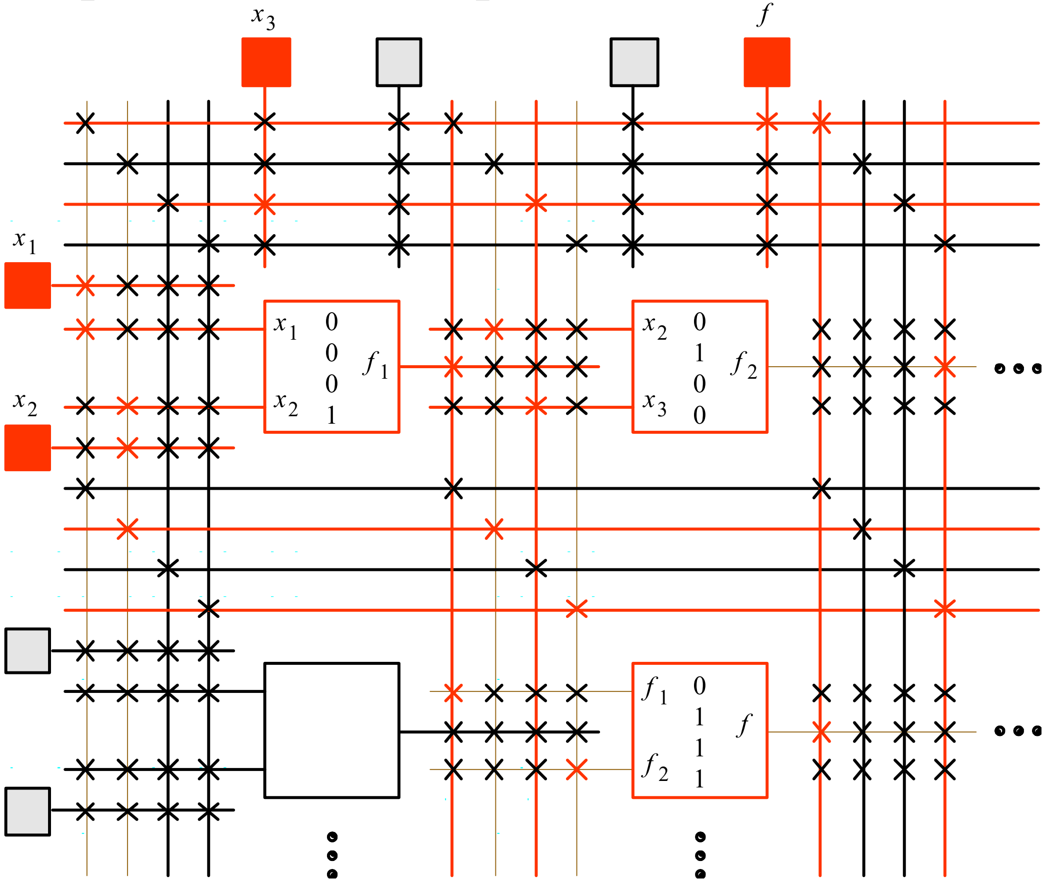


Programming an FPGA

- ISP method is used
- LUTs contain volatile storage cells
  - *None of the other PLD technologies are volatile*
  - *FPGA storage cells are loaded via a PROM when power is first applied*
- The UP2 Education Board by Altera contains a JTAG port, a MAX 7000 CPLD, and a FLEX 10K FPGA
  - *The MAX 7000 CPLD chip is EPM7128SLC84-7*
  - *EPM7 → MAX 7000 family; 128 macrocells; LC84 → 84 pin PLCC package; 7 → speed grade*

Example FPGA

- Use an FPGA with 2 input LUTS to implement the function f = $x_1x_2 + x_2'x_3$



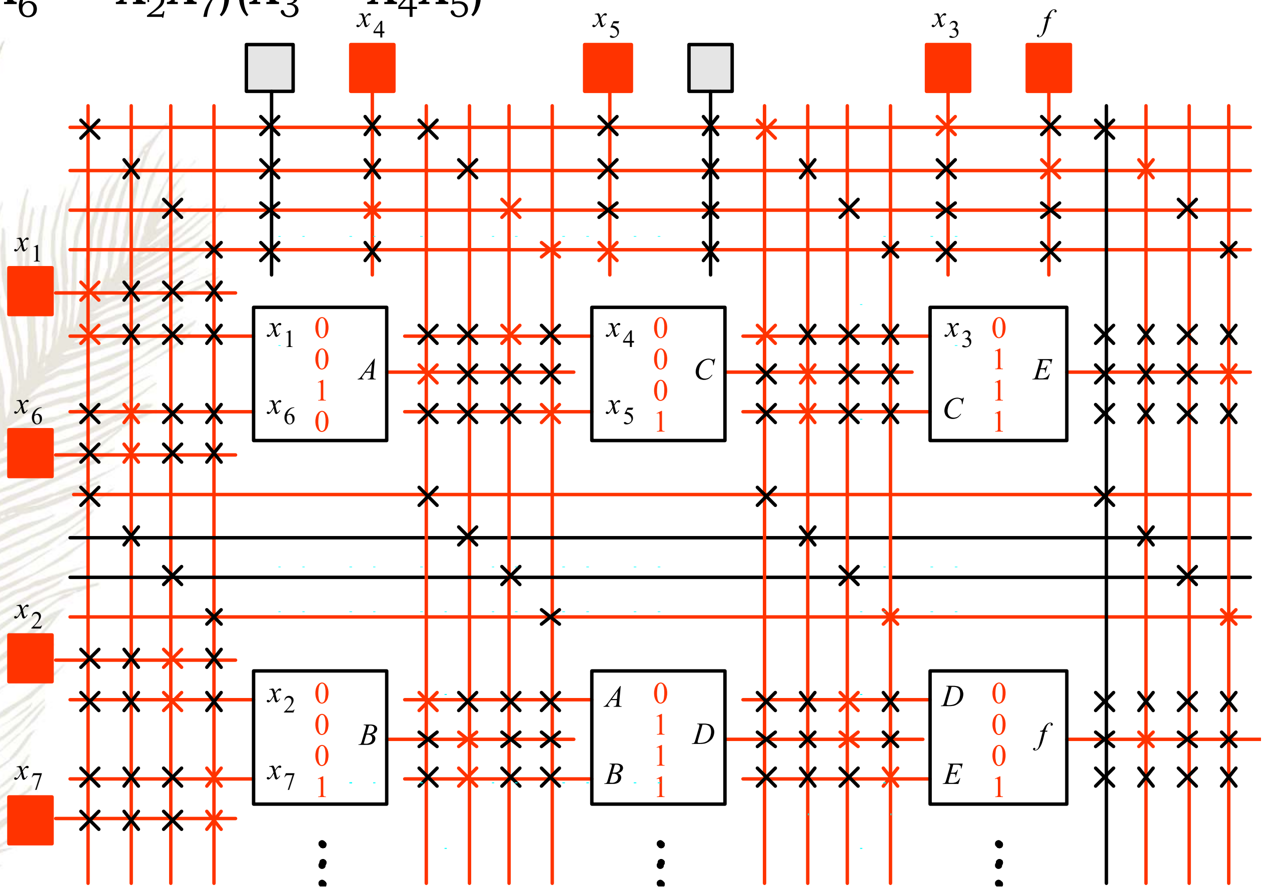
  - $f_1 = x_1x_2$
  - $f_2 = x_2'x_3$
  - $f = f_1 + f_2$

Another Example FPGA

- Use an FPGA with 2 input LUTS to implement the function
- $f = x_1x_3x_6' + x_1x_4x_5x_6' + x_2x_3x_7 + x_2x_4x_5x_7$
  - *Fan-in of expression is too large for FPGA (this was simple to do in a CPLD)*
  - *Factor f to get sub-expressions with max fan-in = 2*
    - $f = x_1x_6'(x_3 + x_4x_5) + x_2x_7(x_3 + x_4x_5)$
      $= (x_1x_6' + x_2x_7)(x_3 + x_4x_5)$
  - *Could use Shannon's expansion instead*
    - Goal is to build expressions out of 2-input LUTs

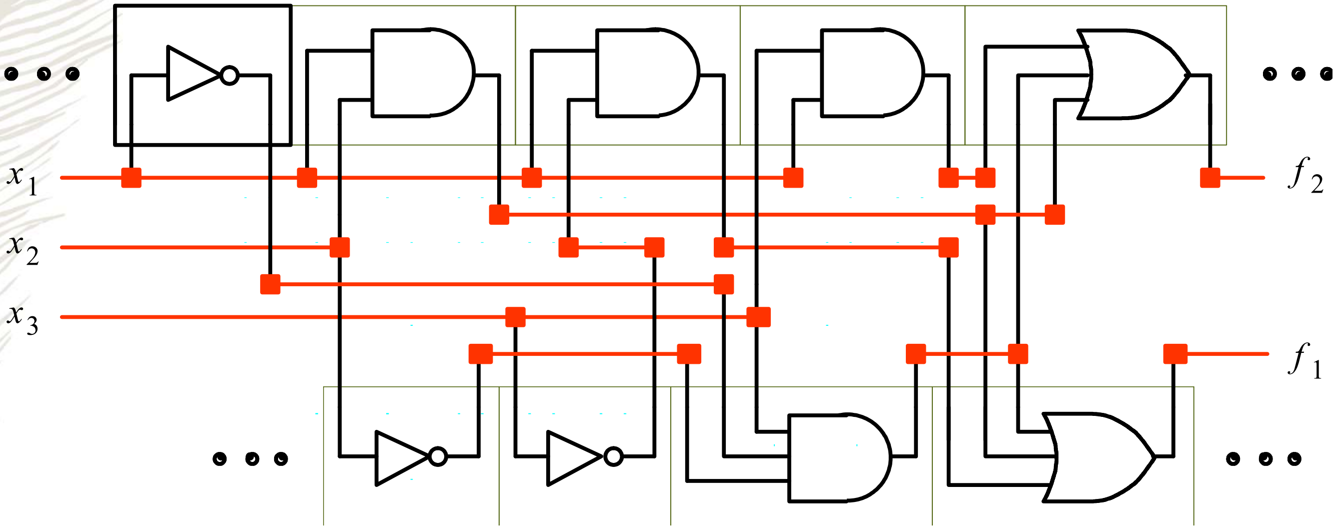
FPGA Implementation

- $f = (x_1x_6' + x_2x_7)(x_3 + x_4x_5)$



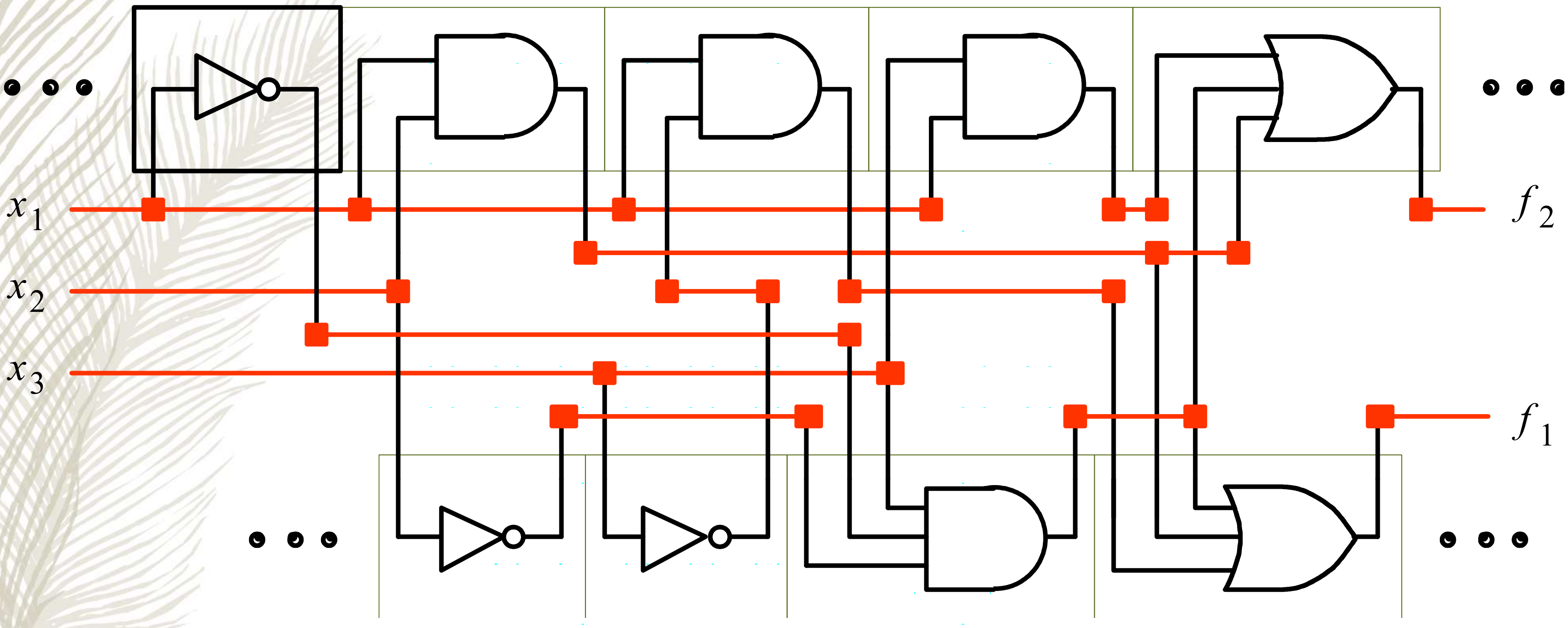
Standard Cells

- Rows of logic gates can be connected by wires in the *routing channels*
  - *Designers (via CAD tools) select prefab gates from a library and place them in rows*
  - *Interconnections are made by wires in routing channels*
    - Multiple layers may be used to avoid short circuiting
    - A hard-wired connection between layers is called a *via*



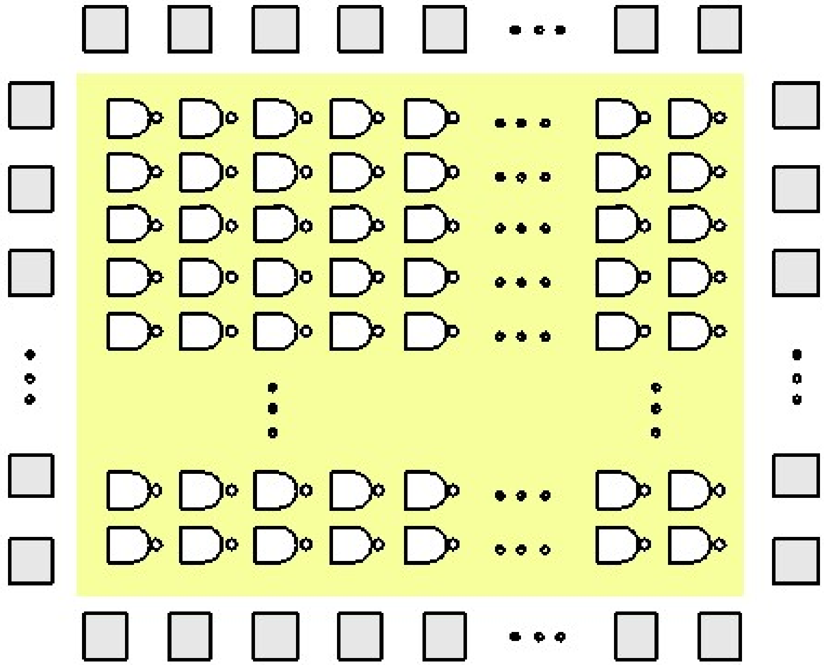
Example: Standard Cells

- $f_1 = x_1x_2 + x_1'x_2'x_3 + x_1x_3'$
- $f_2 = x_1x_2 + x_1'x_2'x_3 + x_1x_3$



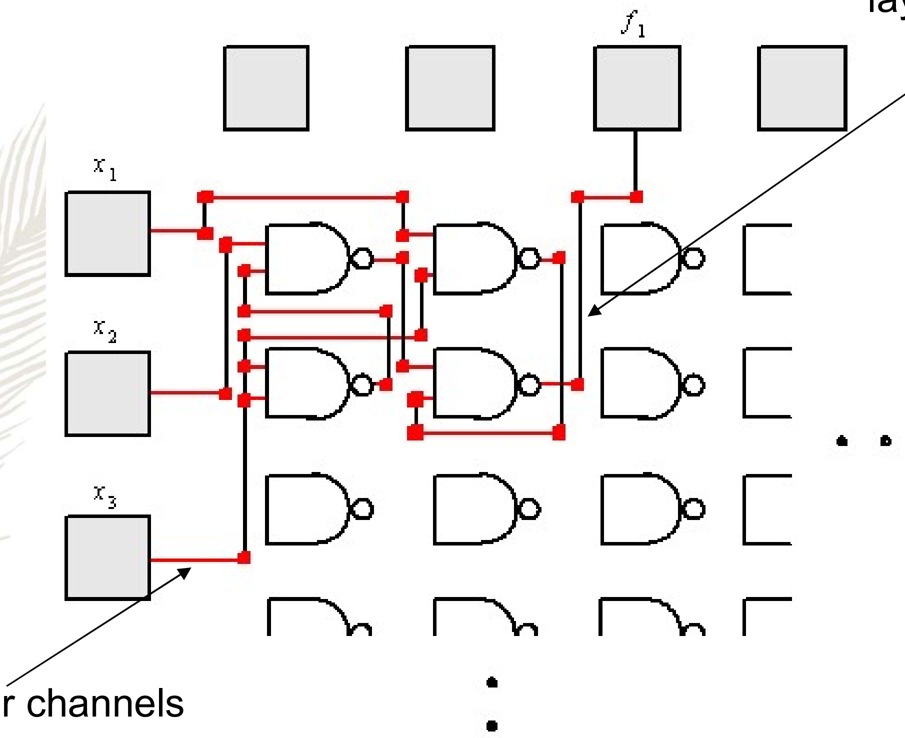
Sea of Gates Gate Array

- A Sea of Gates gate array is just like a standard cell except all gates are of the same type
  - *Interconnections are run in channels and use multiple layers*
  - *Cheaper to manufacture due to regularity*



Example: Sea of Gates

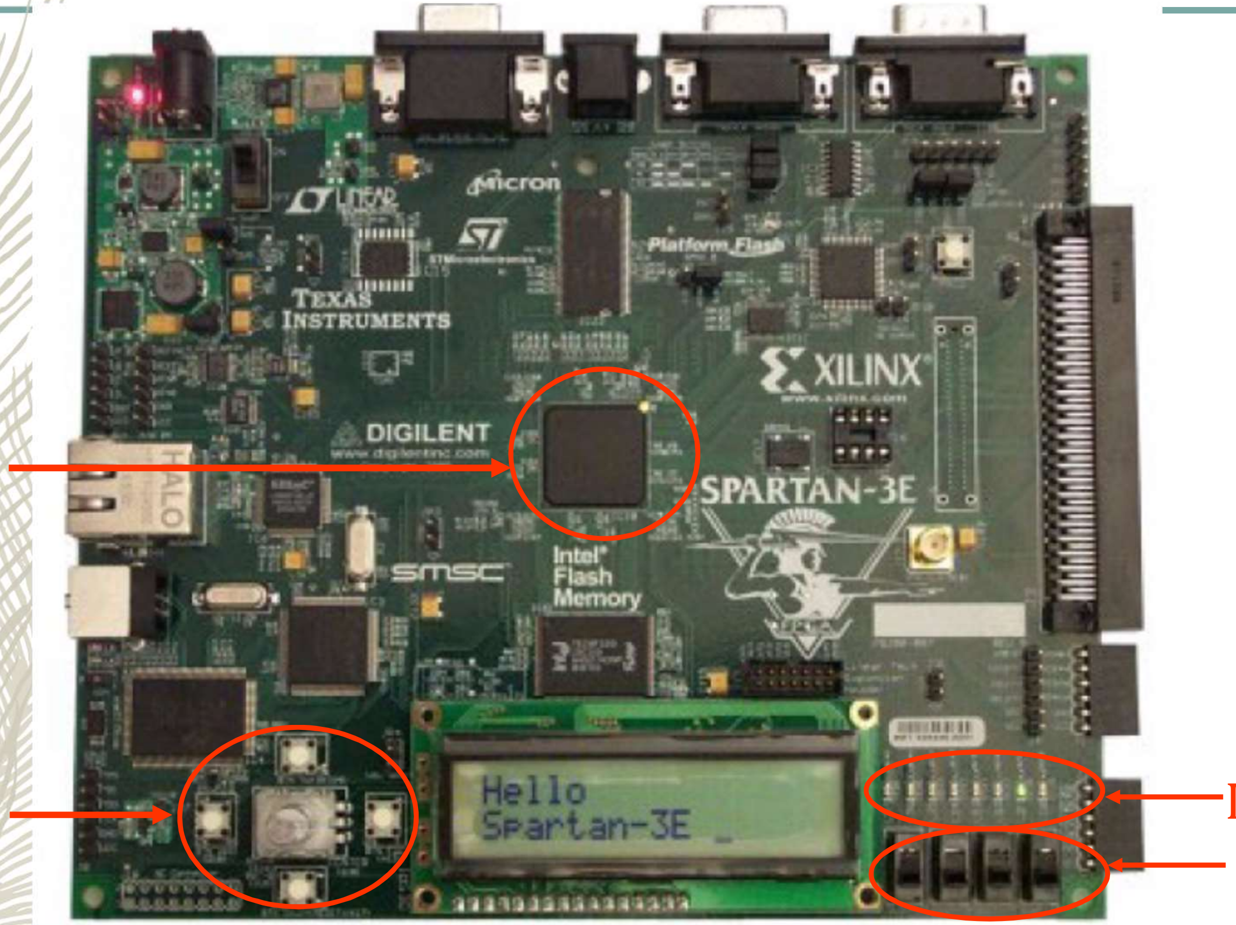
- $f_1 = x_2x_3' + x_1x_3$



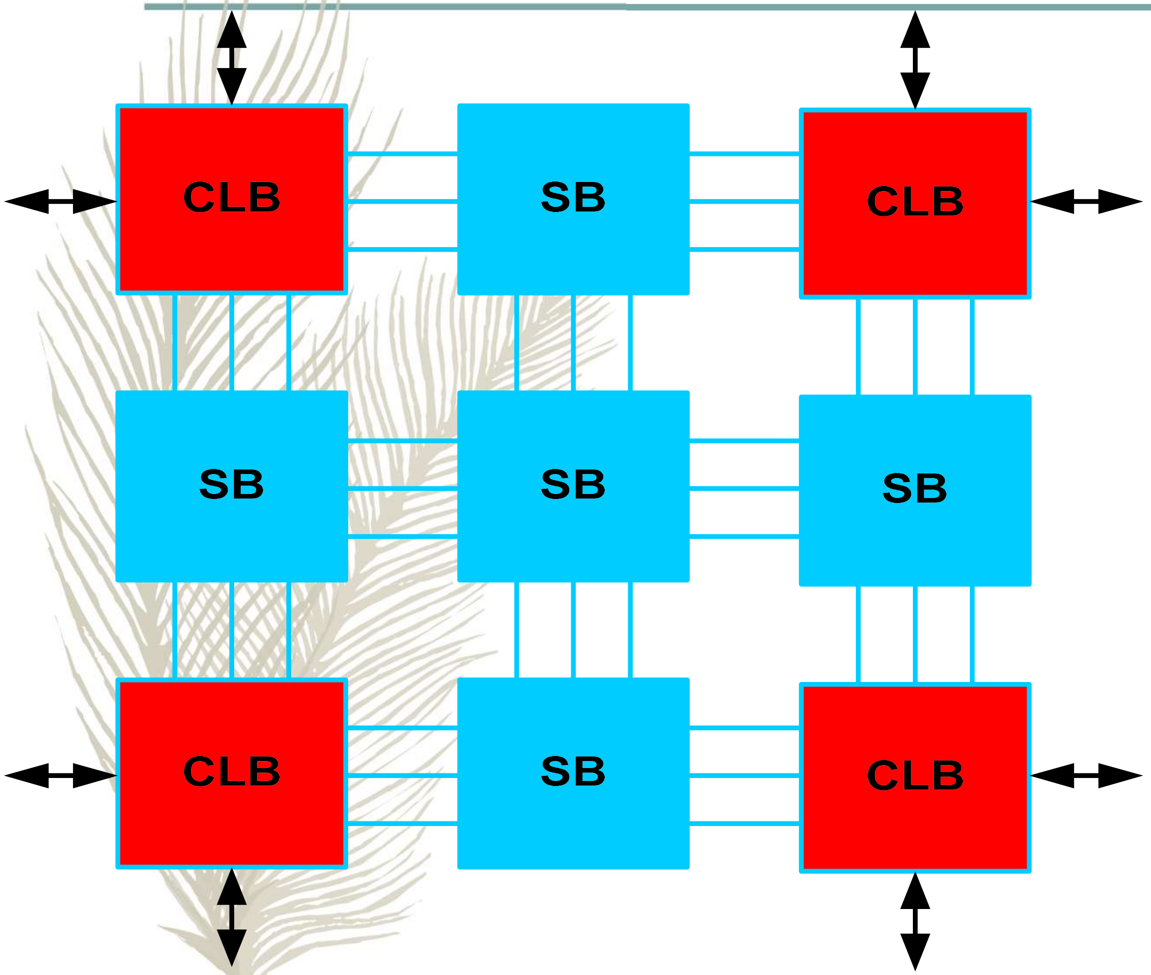
black → bottom layer channels

red → top layer channels

Xilinx Spartan-3E Starter Kit



FPGA structure

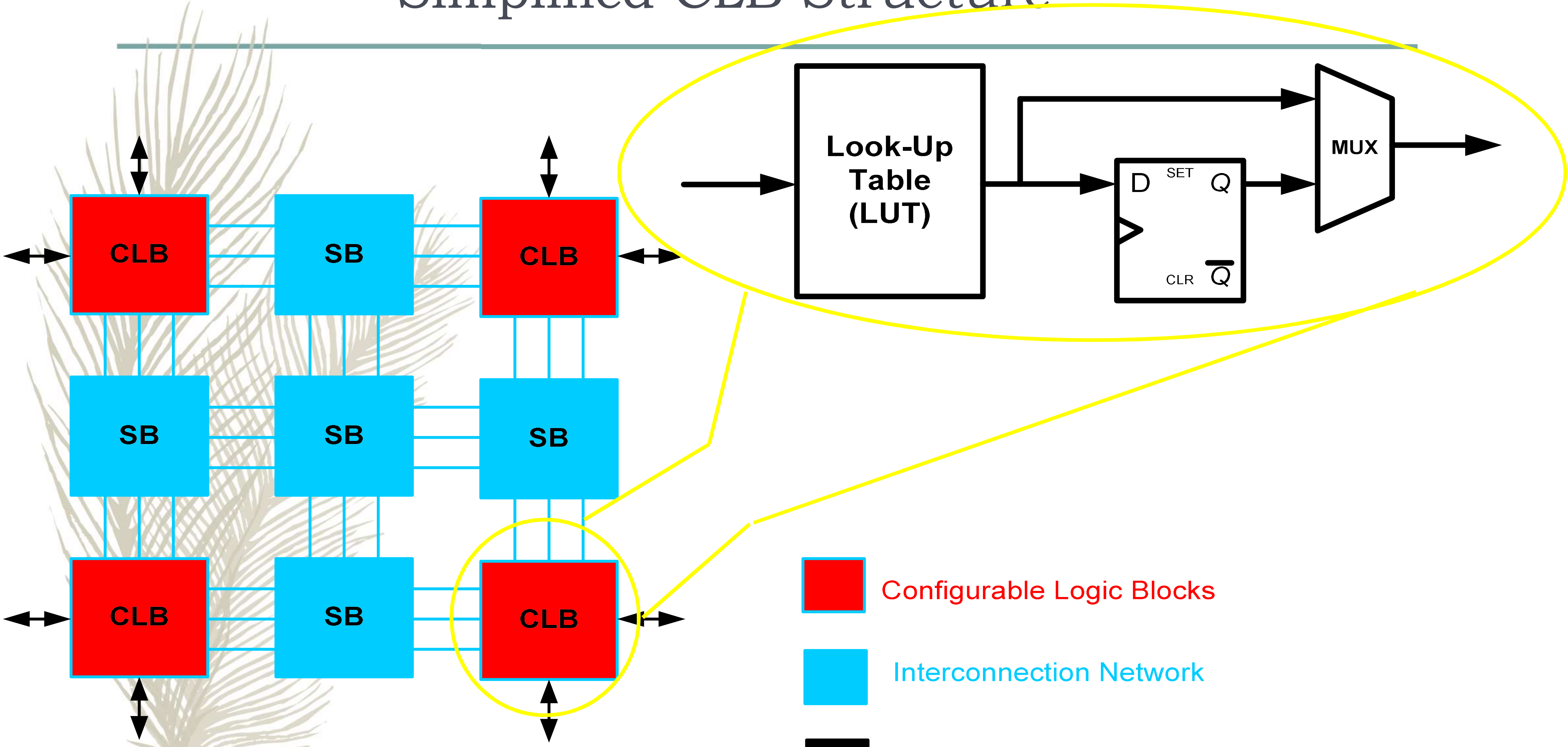


🟥 Configurable Logic Blocks

🟦 Interconnection Network

■ I/O Signals (Pi

Simplified CLB Structure

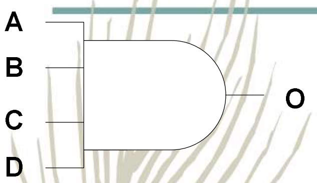


🟥 Configurable Logic Blocks

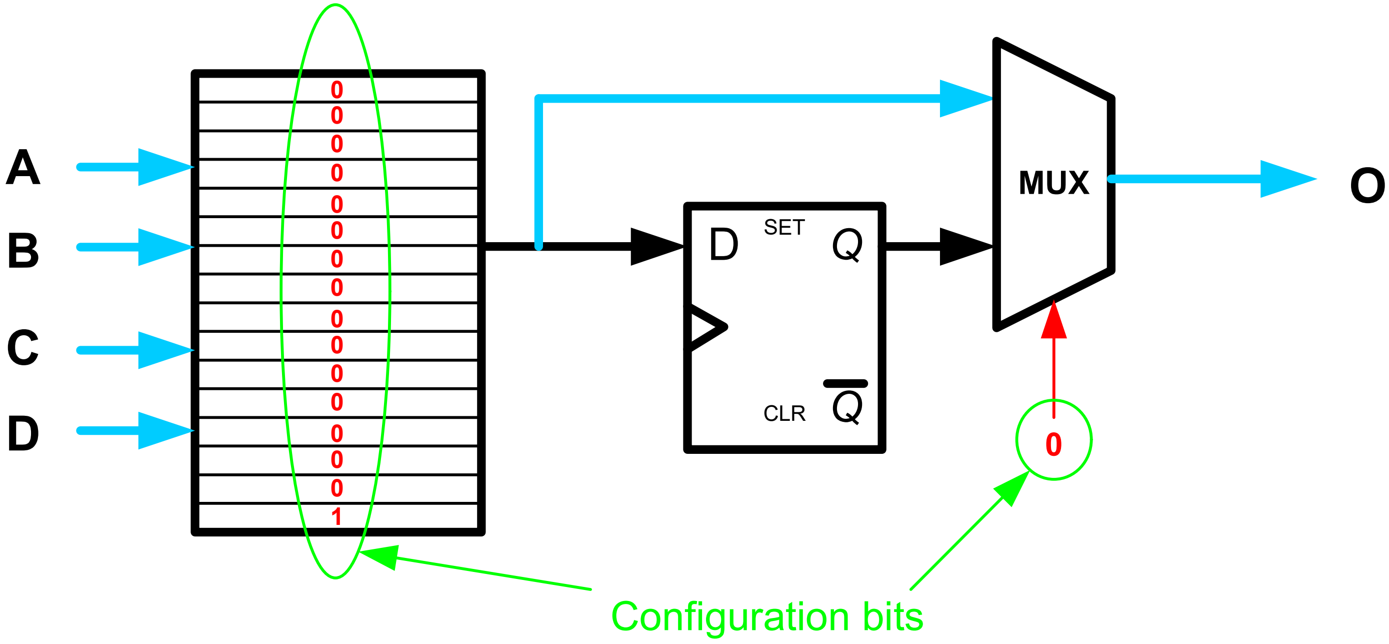
🟦 Interconnection Network

⬛ I/O Signals (Pins)

Example: 4-input AND gate



A	B	C	D	O
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1



FPGA Advantages

- Long time availability
- Can be updated and upgraded at your customer's site
- Extremely short time to market
- Fast and efficient systems
- Performance gain for software applications
- Real time applications
- Massively parallel data processing

FPGA

- FPGA applications:-
  - DSP
  - Software-defined radio
  - Aerospace
  - Defense system
  - ASIC Prototyping
  - Medical Imaging
  - Computer vision
  - Speech Recognition
  - Cryptography
  - Bioinformatics
  - And others.

CMOS TESTING

Topics

- Need for testing
- Test principles
- Design strategies for test
- Chip level test techniques
- System level test techniques

Need for Testing

- The need of testing is to find out errors in the application. The good reasons of testing are

  1) Quality Assurance.

  2) Verification and validating the product/application before it goes live in the market.

  3) Defect free and user friendly.

  4) Meets the requirements.

- Testing is one of the most expensive parts of chips
  - Logic verification accounts for > 50% of design effort for many chips
  - Debug time after fabrication has enormous opportunity cost
  - Shipping defective parts can sink a company
  - Decreasing feature size increases probability of defects during manufacturing process.
  - imperfections in chip fabrication may lead to manufacturing defects

  The manufacturing yield (Y) depends on the used technology,the silicon area and the layout design. Y = no of acceptable parts / total no of fabricated parts

- Example: Intel FDIV bug (1994)
  - Logic error not caught until > 1M units shipped * Recall cost $450M (!!!)

Rule of Ten

- ➢ Rule of Ten: cost to detect faulty IC increases by an order of magnitude as we move from:
- ❖ device →PCB →system →field operation– Testing performed at all of these levels
- ➢ Cost to detect a fault (per chip)
  - ❖ Wafer: $0.01-$0.1
  - ❖ Packaged chip: $0.1-$1
  - ❖ Board: $1-$10
  - ❖ System: $10-$100
  - ❖ Field: $100-1000

Role of Testing

Detection: Determination whether or not the device under test (DUT) has some fault.

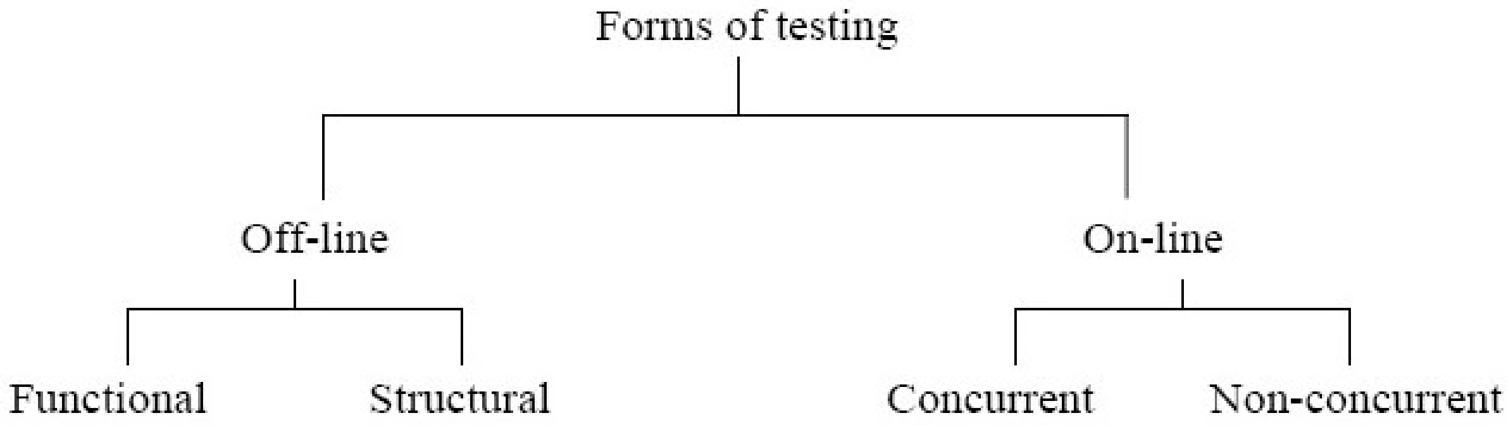
Diagnosis: Identification of a specific fault that is present on DUT.

Device characterization: Determination and correction of errors in design and/or test procedure.

Failure mode analysis (FMA): Determination of manufacturing process errors that may have caused defects on the DUT.

- **Defects :** are circuit failures and malfunctions due to the manufacturing process(eg:short circuits opens.
- **Faults :**model the influence of defects on the circuit operation (e.g. a line (node) is permanently stuck-at“1” or “0”).
- **Errors:** are the incorrect logic responses of the circuit under the presence of faults.

Classification of tests



Forms of testing

Off-line

On-line

Functional

Structural

Concurrent

Non-concurrent

Off Chip and On Chip Testing

Off chip testing: The test procedures are applied by external to the chip test equipment's (Automatic Test Equipment–ATEor Tester).

On chip testing: Embedded, on-chip, resources are provided in order to support the testing procedures.

❖ **On-line testing:** Testing procedures are applied in the field of operation. Concurrent testing: Testing is performed concurrently with the circuit operation in the field, during the normal mode.

❖ Periodic testing: Testing is performed periodically, during idle times of the circuit operation.

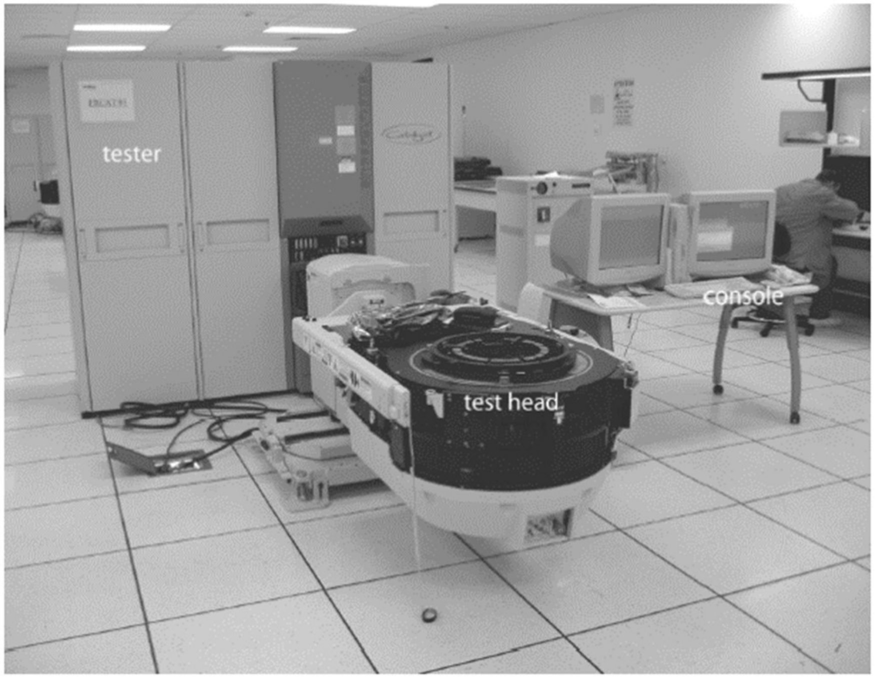
•**Off-line testing:** Testing procedures are applied out of the VLSI Testing field of operation, usually after fabrication (manufacturing testing).

Logic Verification

- Does the chip simulate correctly?
  - Usually done at HDL level
  - Verification engineers write test bench for HDL
    - *Can’t test all cases*
    - *Look for corner cases*
    - *Try to break logic design*
- Ex: 32-bit adder
  - Test all combinations of corner cases as inputs:
    - *0, 1, 2, 231-1, -1, -231, a few random numbers*
- Good tests require ingenuity

Manufacturing Test

- A speck of dust on a wafer is sufficient to kill chip
- Yield of any chip is < 100%
  - Must test chips after manufacturing before delivery to customers to only ship good parts
- Manufacturing testers are very expensive
  - Minimize time on tester
  - Careful selection of
  - test vectors



Defects: layer-to-layer shorts, discontinuous wires

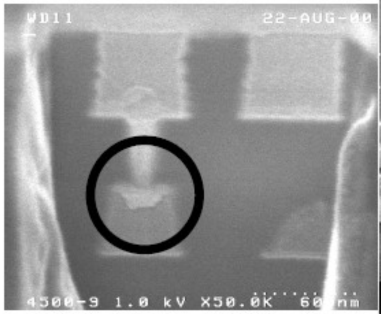
thin-oxide shorts to substrate or well

Faults: nodes shorted to power or ground (stuck-at)

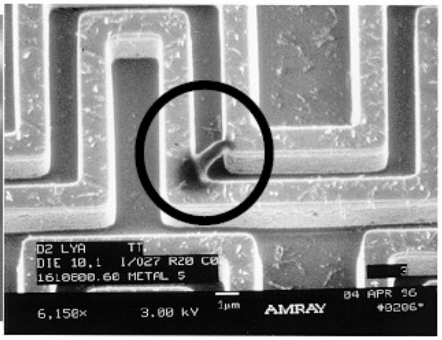
nodes shorted to each other (bridging)

inputs floating, outputs disconnected (stuck-open)

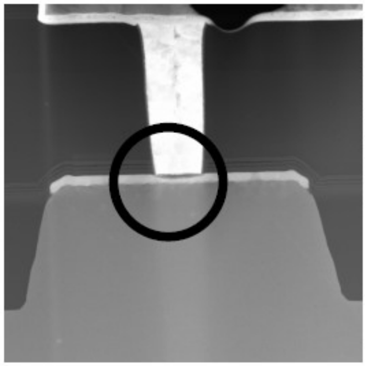
Manufacturing Failures



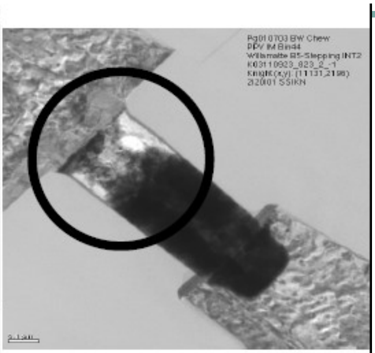
Metal 1 Shelving



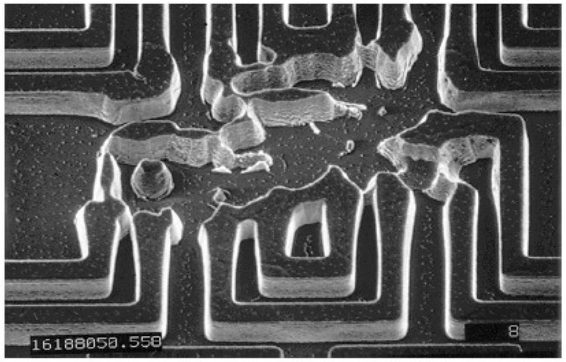
Metal 5 film particle (bridging defect)



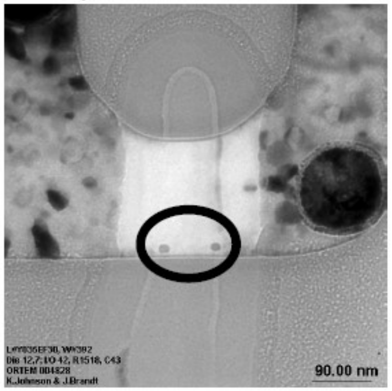
Open defect



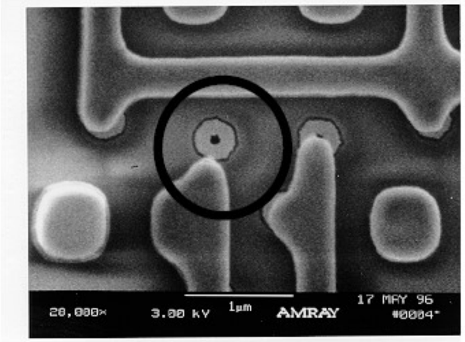
Spongy Via2 (Infant mortality)



Metal 5 blocked etch (patterning defect)



Spot defects
“Co” Defect under Gate



Metal 1 missing pattern (open at contact)

SEM images courtesy Intel Corporation

Fault types and Models

- Errors :– Permanent– Intermittent– Transient
- Faults :– Physical– Logical

The physical defects can cause electrical faults and logical faults.

The **electrical faults** include: Shorts (bridging faults) & opens-Transistor stuck-on, stuck-open-Resistive shorts and opens-Excessive change in threshold voltage-Excessive steady-state currents

The **logical faults** include: Logical struck-at-0 or struck-at-1-Slower transition-AND-bridging, OR bridging

Electrical Faults : Stuck-On, Stuck-Open

Test Process

- The testing problem

Given a set of faults in the circuit under test (or device under test), how do we obtain a certain(small) number of test patterns which guarantees a certain (high) fault coverage?

 Test process

 What faults to test? (***fault modeling***)

 How are test pattern obtained? (***test pattern generation***)

 How is test quality (fault coverage) measured?

(***fault simulation***)?

 How are test vectors applied and results evaluated?

(***ATE/BIST***)

Why Model Faults ?

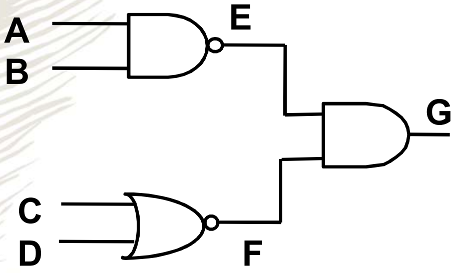
- Identifies target faults
  - Model faults most likely to occur
- Limits the scope of test generation
  - Create tests only for the modeled faults
- Makes analysis possible
  - Associate specific defects with specific test patterns
- Makes test effectiveness measurable by experiments
  - Fault coverage can be computed for specific test patterns to reflect its effectiveness

Fault Nature & Fault Duration

- **Logical fault** :One that causes the logic function of a circuit element to be changed to some other function
- **Parametric fault** : One that alters the magnitude of a circuit parameter, causing a change in some factor such as resistance, capacitance, current, etc.
- **Delay fault** : One that relates to circuit delays such as slow gates, usually affecting the timing of the circuit, which may cause hazards, or performance degradation, etc.
- **Permanent fault** : A lasting fault that is continuous and stable, whose nature does not change before, during, and after testing. E.g., a broken wire, an incorrect bonding, etc.
  - A.k.a hard fault or solid fault
- **Temporary fault :** A fault that is present only part of the time, occurring at random moments and affecting the system for finite, but unknown, intervals of time
  - Transition fault : Caused by environmental conditions, e.g., cosmic rays, alpha particle,etc. A.k.a. soft error in RAMs
  - Intermittent fault : Caused by non-enviornmental conditions, e.g., marginal values of component parameters, wear-out, or critical timing

Fault Modeling

- The effects of physical defects
- Most commonly used fault model: Single stuck-at fault



A s-a-1 B s-a-1 C s-a-1 D s-a-1
A s-a-0 B s-a-0 C s-a-0 D s-a-0

E s-a-1 F s-a-1 G s-a-1
E s-a-0 F s-a-0 G s-a-0

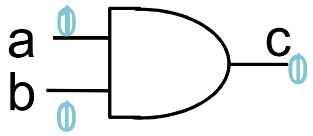
14 faults

- **Other fault models:**
  - Break faults, Bridging faults, Transistor stuck-open faults, Transistor stuck-on faults, Delay faults

Fault Coverage (FC)

$$FC = \frac{\#\text{ faults detected}}{\#\text{ faults in fault list}}$$

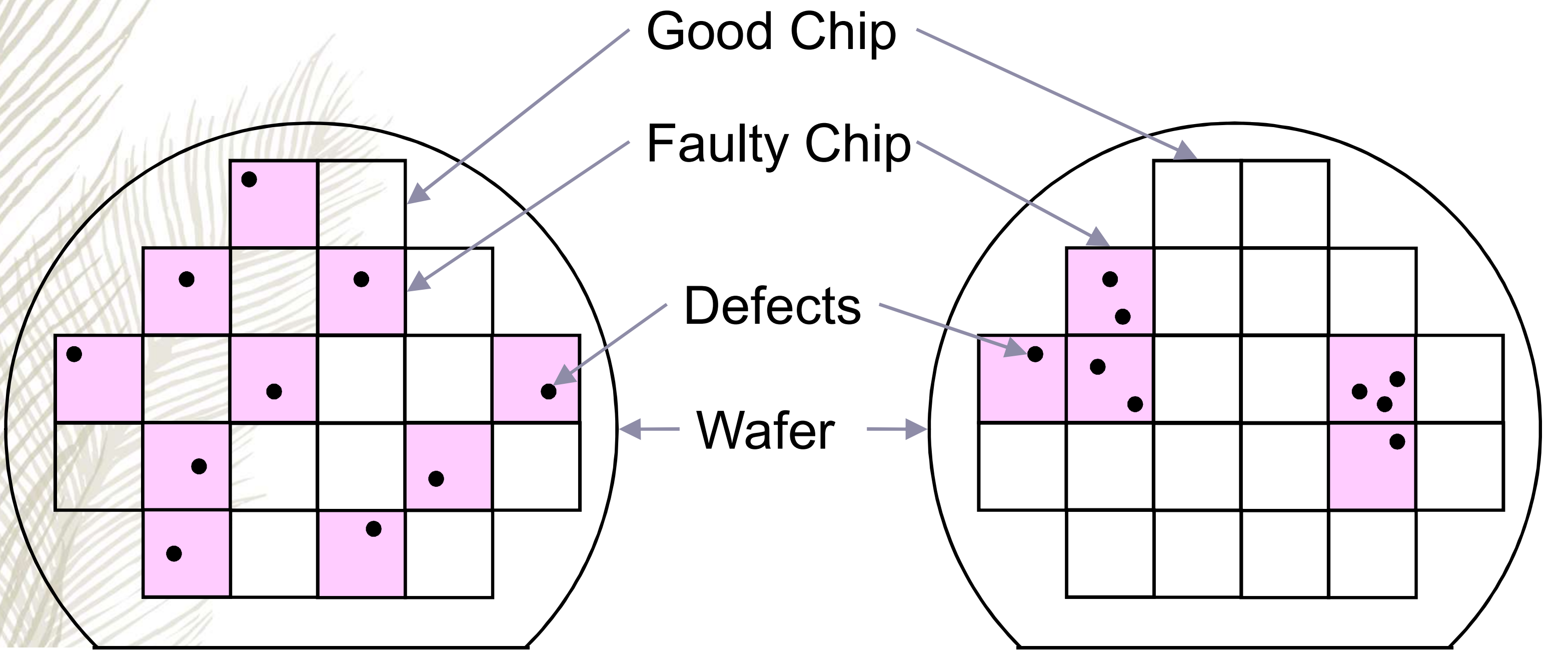
Example:



6 stuck-at faults
(a_0,a_1,b_0,b_1,c_0,c_1)

Test	faults detected	FC
{(0,0)}	c_1	16.67%
{(0,1)}	a_1,c_1	33.33%
{(1,1)}	a_0,b_0,c_0	50.00%
{(0,0),(1,1)}	a_0,b_0,c_0,c_1	66.67%
{(1,0),(0,1),(1,1)}	all	100.00%

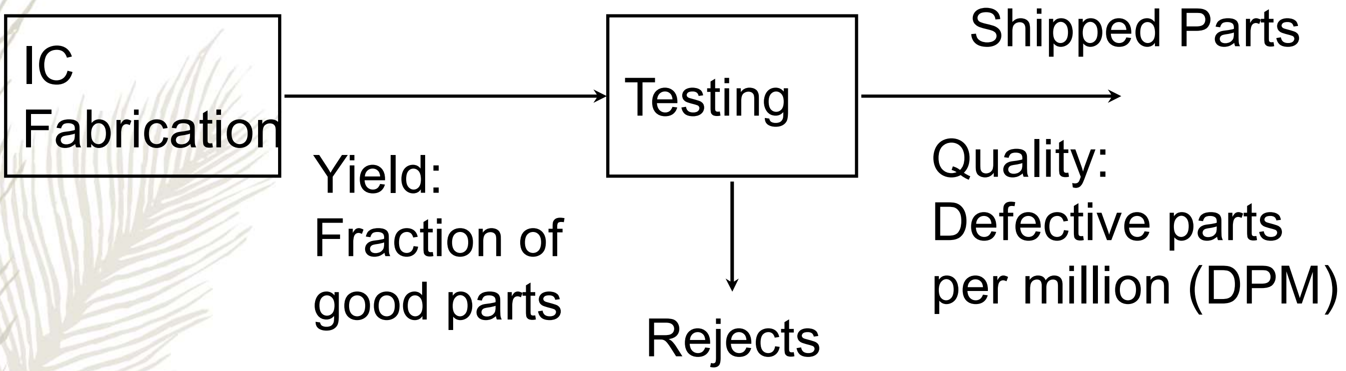
Wafer Yield (Chip Yield, Yield)



Wafer yield = 12/22 = 0.55

Wafer yield = 17/22 = 0.77

Testing and Quality


IC Fabrication
Testing
Shipped Parts
Yield: Fraction of good parts
Rejects
Quality: Defective parts per million (DPM)

- Quality of shipped parts is a function of yield Y and the test (fault) coverage T
- Defect level (DL, reject rate in textbook): fraction of shipped parts that are defective

Defect Level, Yield & Fault Coverage

$$DL \cong 1 - Y^{(1-T)}$$

DL: defect level

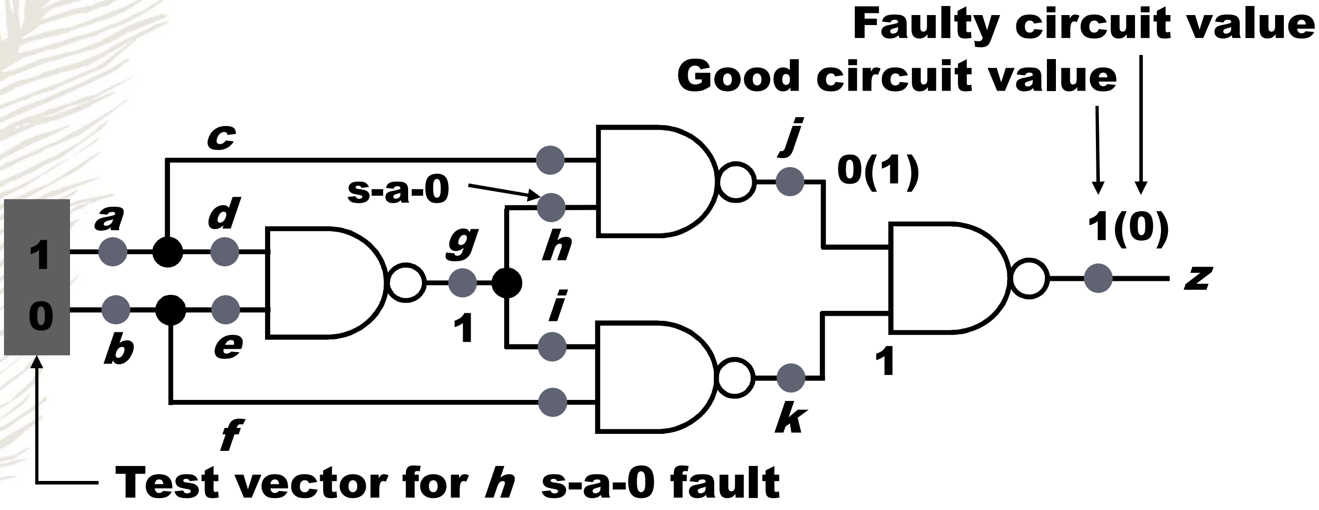
Y: yield

T: fault coverage

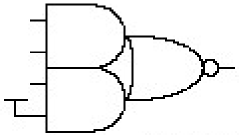
Yield (Y)	Fault Coverage (T)	DPM (DL)
50%	90%	67,000
75%	90%	28,000
90%	90%	10,000
95%	90%	5,000
99%	90%	1,000
90%	90%	10,000
90%	95%	5,000
90%	99%	1,000
90%	99.9%	100

Single Stuck-at Fault

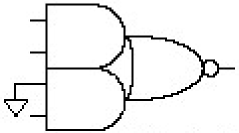
- Three properties define a single stuck-at fault
  - *Only one line is faulty*
  - *The faulty line is permanently set to 0 or 1*
  - *The fault can be at an input or output of a gate*
- Example: XOR circuit has 12 fault sites (●) and 24 single stuck-at faults



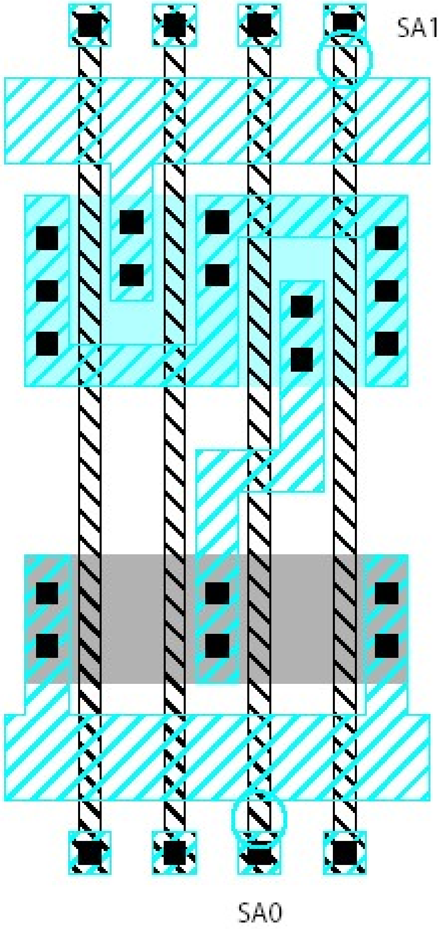
Examples



Stuck-At-1
SA1 Fault



Stuck-At-0
SA0 Fault



Design Strategies for Test:

- Observability: ease of observing a node by watching external output pins of the chip
- Controllability: ease of forcing a node to 0 or 1 by driving input pins of the chip

- Combinational logic is usually easy to observe and control
- Finite state machines can be very difficult, requiring many cycles to enter desired state
  - Especially if state transition diagram is not known to the test engineer

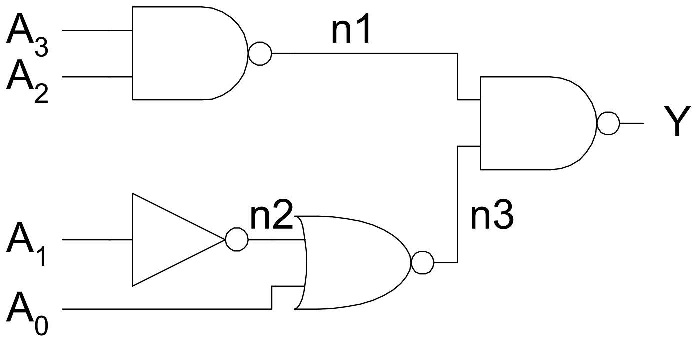
Test Pattern Generation

- Manufacturing test ideally would check every node in the circuit to prove it is not stuck.
- Apply the smallest sequence of test vectors necessary to prove each node is not stuck.

- Good observability and controllability reduces number of test vectors required for manufacturing test.
  - Reduces the cost of testing
  - Motivates design-for-test

Test Example

	SA1	*SA0*
➢ A3	{0110}	{1110}
➢ A2	{1010}	{1110}
➢ A1	{0100}	{0110}
➢ A0	{0110}	{0111}
➢ n1	{11	{0110}
➢ n2	{0110}	{0100}
➢ n3	{0101}	{0110}
➢ Y	{0110}	{1110}



- ➢ Minimum set: {0100, 0101, 0110, 0111, 1010, 1110}

Design for Test

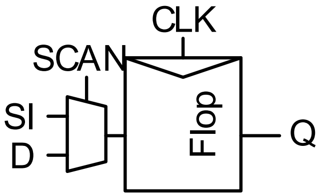
- Design the chip to increase observability and controllability

- If each register could be observed and controlled, test problem reduces to testing combinational logic between registers.

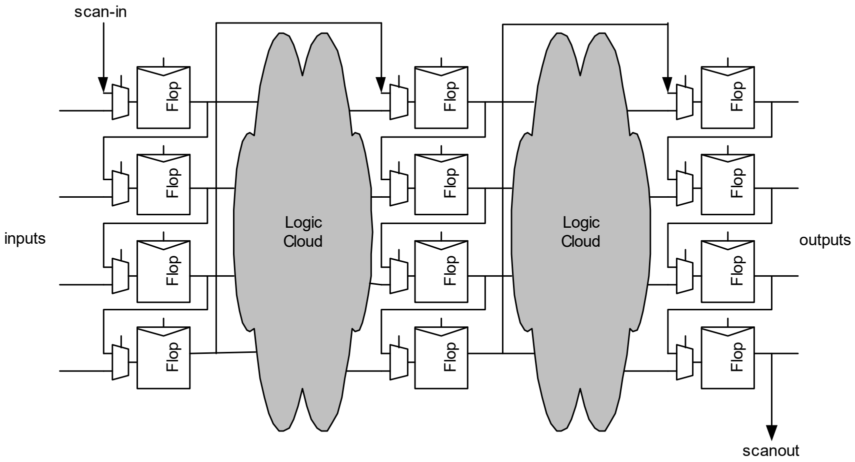
- Better yet, logic blocks could enter test mode where they generate test patterns and report the results automatically.

Scan

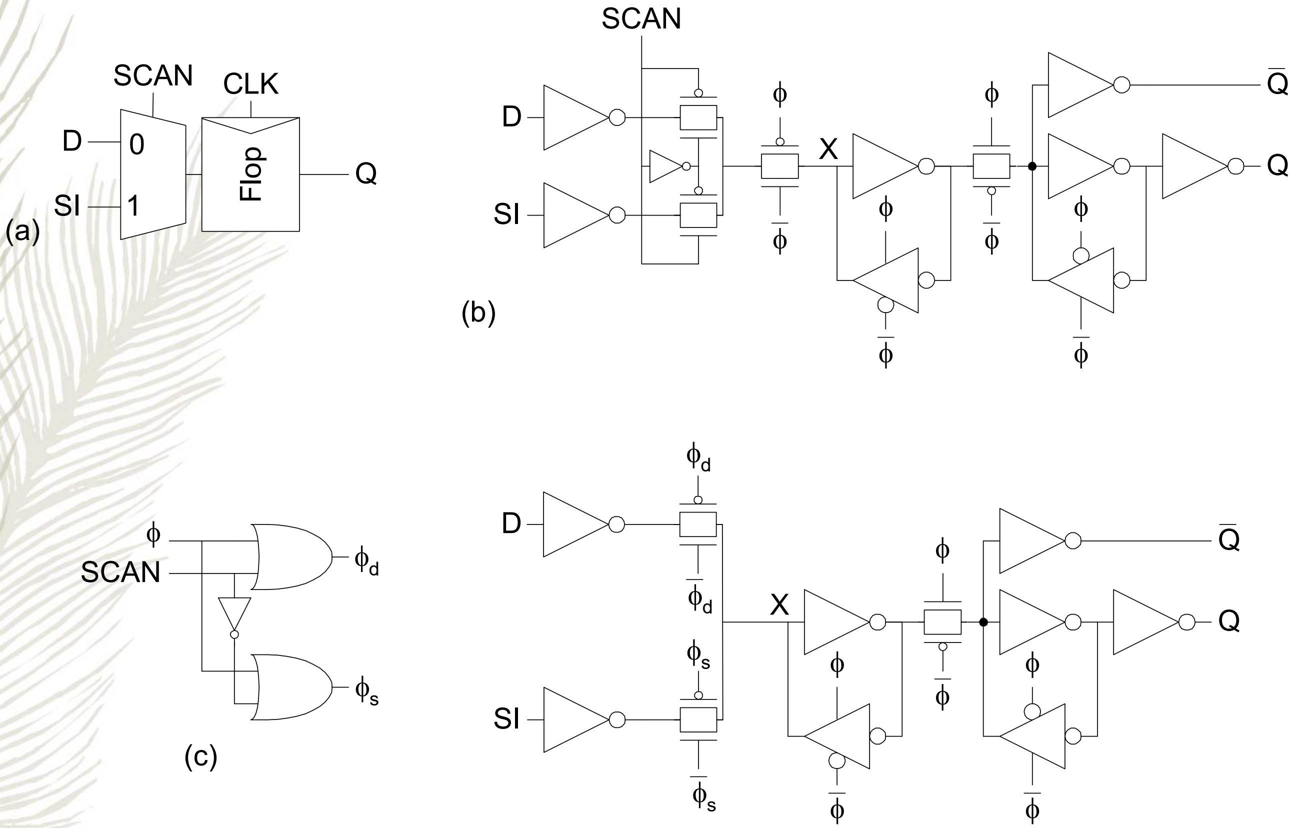
- Convert each flip-flop to a scan register
  - Only costs one extra multiplexer
- Normal mode: flip-flops behave as usual
- Scan mode: flip-flops behave as shift register



- Contents of flops
- can be scanned
- out and new
- values scanned
- in



Scannable Flip-flops



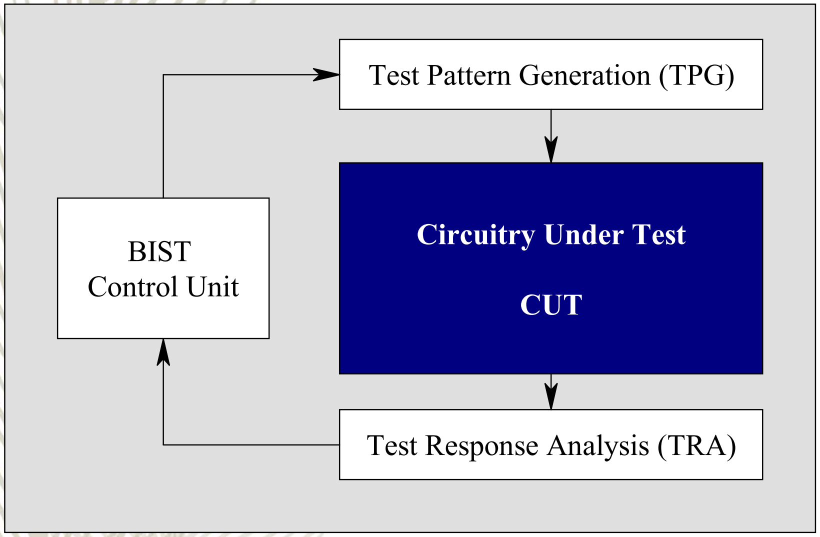
ATPG

- Test pattern generation is tedious
- Automatic Test Pattern Generation (ATPG) tools produce a good set of vectors for each block of combinational logic
- Scan chains are used to control and observe the blocks
- Complete coverage requires a large number of vectors, raising the cost of test
- Most products settle for covering 90+% of potential stuck-at faults

Built-in Self-test

- Built-in self-test lets blocks test themselves
  - Generate pseudo-random inputs to comb. logic
  - Combine outputs into a syndrome
  - With high probability, block is fault-free if it produces the expected syndrome

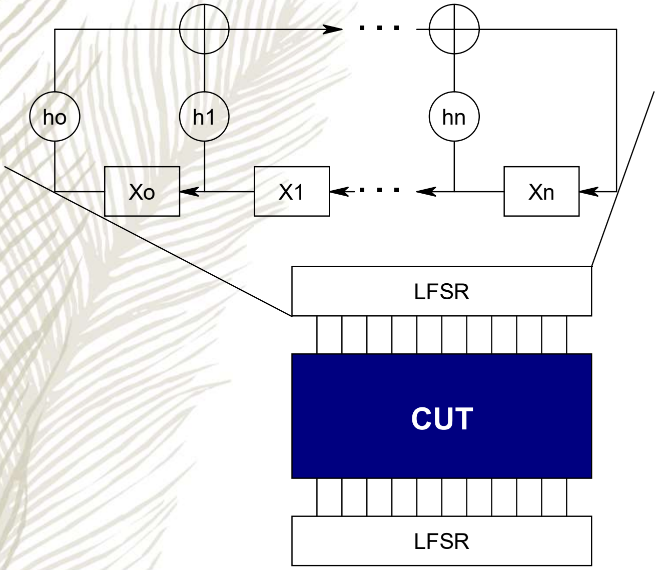
General Architecture of BIST



- BIST components:
  - Test pattern generator (TPG)
  - Test response analyzer (TRA)
- TPG & TRA are usually implemented as linear feedback shift registers (LFSR)
- Two widespread schemes:
  - test-per-scan
  - test-per-clock

Pattern Generation

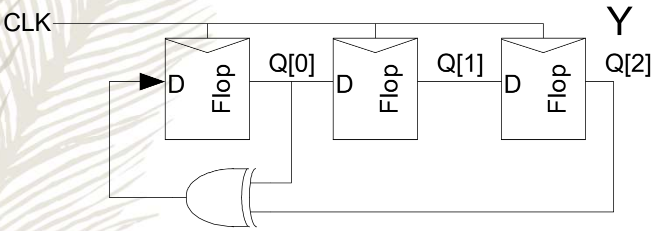
Pseudorandom Test generation by LFSR:



- Using special LFSR registers
- Several proposals:
  - BILBO
  - CSTP
- Main characteristics of LFSR:
  - polynomial
  - initial state
  - test length

PRSG

- Linear Feedback Shift Register
  - Shift register with input taken from XOR of state
  - Pseudo-Random Sequence Generator

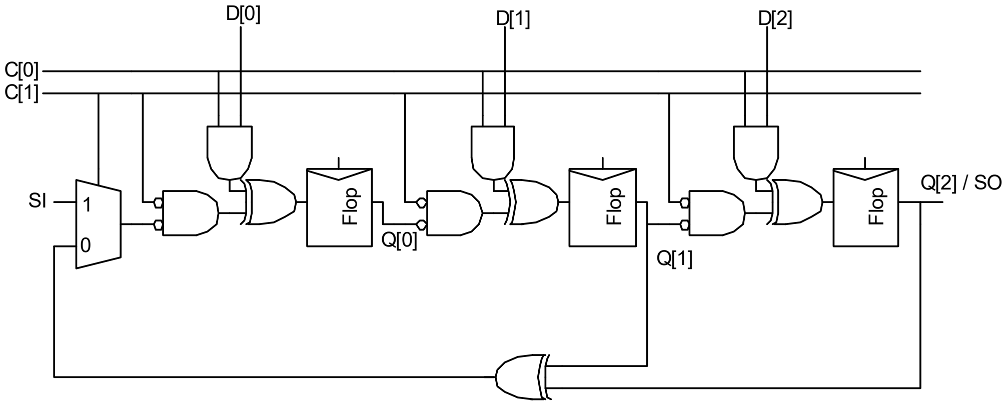


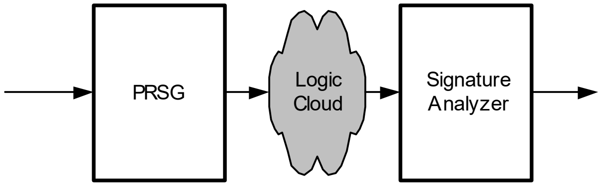
Flops reset to 111

Step	Y
0	111
1	110
2	101
3	010
4	100
5	001
6	011
7	111 (repeats)

BILBO

- ➢ Built-in Logic Block Observer
  - ➢ Combine scan with PRSG & signature analysis



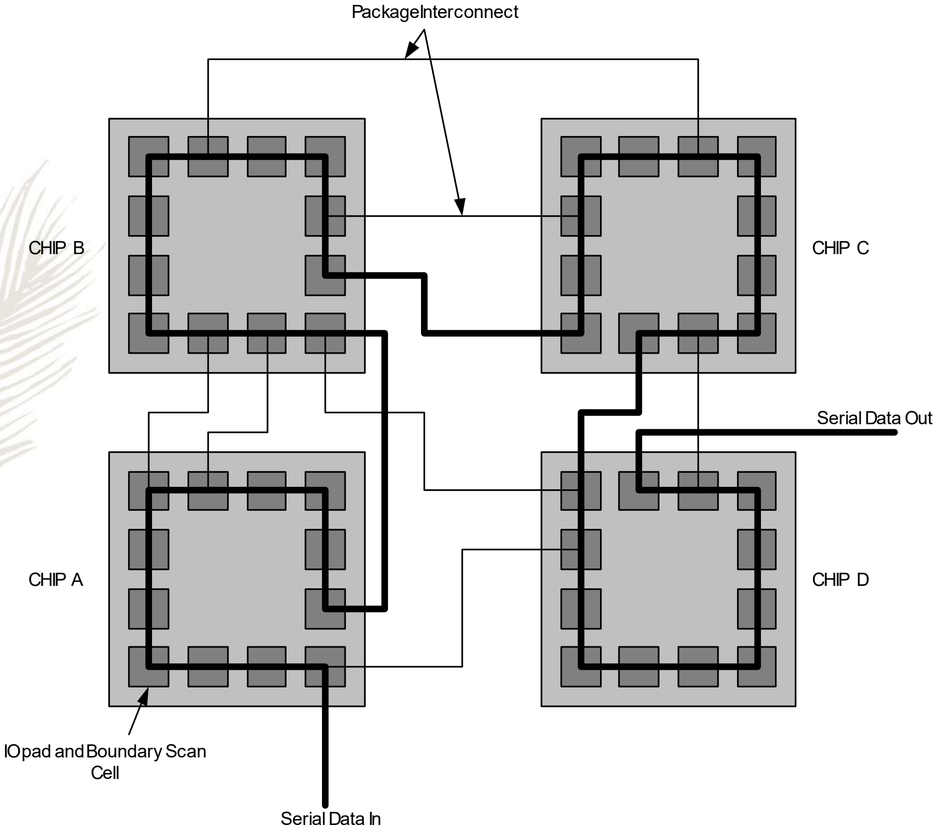


MODE	C[1]	C[0]
Scan	0	0
Test	0	1
Reset	1	0
Normal	1	1

Boundary Scan

- Testing boards is also difficult
  - Need to verify solder joints are good
    - *Drive a pin to 0, then to 1*
    - *Check that all connected pins get the values*
- Through-hold boards used “bed of nails”
- SMT and BGA boards cannot easily contact pins
- Build capability of observing and controlling pins into each chip to make board test easier

Boundary Scan Example



Boundary Scan Interface

- Boundary scan is accessed through five pins
  - TCK: test clock
  - TMS: test mode select
  - TDI: test data in
  - TDO: test data out
  - TRST*: test reset (optional)
- Chips with internal scan chains can access the chains through boundary scan for unified test strategy.

Some Definitions

- **LFSR** – Linear feedback shift register, hardware that generates pseudo-random pattern sequence
- **BILBO** – Built-in logic block observer, extra hardware added to flip-flops so they can be reconfigured as an LFSR pattern generator or response compacter, a scan chain, or as flip-flops
- **Exhaustive testing** – Apply all possible 2n patterns to a circuit with n inputs
- **Pseudo-exhaustive testing** – Break circuit into small, overlapping blocks and test each exhaustively
- **Pseudo-random testing** – Algorithmic pattern generator that produces a subset of all possible tests with most of the properties of randomly-generated patterns

Thank you………………..