CMPT-150

Introduction to Computer Design

SFU, Harbour Centre, Spring 2007

Lecture 9: Feb. 6, 2007

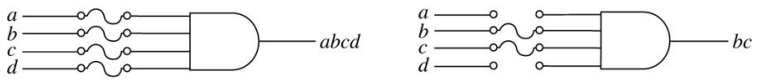
- Programmable Logic Devices (PLDs)
  - Read Only Memory (ROM)
  - Programmable Array Logic (PAL)
  - Programmable Logic Array (PLA)
- Implementing Boolean function using PLDs
- Wired-AND and Wired-OR buses

Review: Chip Design Styles

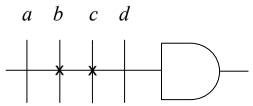
- Full custom - the entire design of the chip down to the smallest detail of the layout is performed
  - Expensive
  - Justifiable only for dense, fast chips with high sales volume
- Standard cell - blocks have been design ahead of time or as part of previous designs
  - Intermediate cost
  - Less density and speed compared to full custom
- **Gate array** - regular patterns of gate transistors that can be used in many designs built into chip - only the interconnections between gates are specific to a design
  - Lowest cost
  - Less density compared to full custom and standard cell

What does "Programmable" Logic Mean?

- Programming = specifying the interconnections between gates, usually within "arrays" of OR and AND gates.
- Programming Technologies:
  - Fuse



Every input is connected through a "fuse" to the gate
By "blowing" the fuses connecting a and d we implement bc.*
Notation:



* We assume a disconnected input signal carries:
- "1" for an AND gate
- "0" for an OR gate

What does "Programmable" Logic Mean? (cont')

- Other programming technologies:
  - Antifuse
  - Mask programming
  - Single bit storage element
- Characterization of programming technologies:
  - Permanent – once set cannot be changed
  - Reprogrammable
    - Volatile – programming lost once electric power is off
    - Non-volatile
      - Erasable
      - Electrically erasable
      - Flash (as in Flash Memory)

Why Programmable Logic?

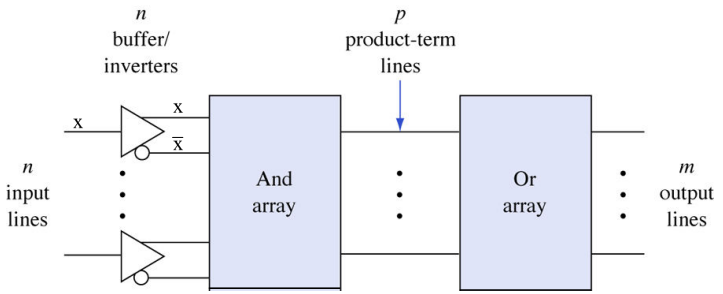
- Facts:
  - It is most economical to produce an IC in large volumes
  - Many designs required only small volumes of ICs
- Need an IC that can be:
  - Produced in large volumes
  - Handle many designs required in small volumes
- A programmable logic part can be:
  - made in large volumes
  - programmed to implement large numbers of different low-volume designs

Programmable Logic - Additional Advantages

- Many programmable logic devices are *field-programmable*, i. e., can be programmed outside of the manufacturing environment
- Most programmable logic devices are *erasable* and *reprogrammable*.
  - Allows "updating" a device or correction of errors
  - Allows reuse the device for a different design - the ultimate in re-usability!
  - Ideal for course laboratories
- Programmable logic devices can be used to prototype design that will be implemented for sale in regular ICs.
  - Complete Intel Pentium designs were actually prototype with specialized systems based on large numbers of VLSI programmable devices!

Types of PLDs

General structure of a PLD:



- Types of PLDs:

Device	And-array	Or-array
ROM	Fixed	Programmable
PAL	Programmable	Fixed
PLA	Programmable	Programmable

Read Only Memory

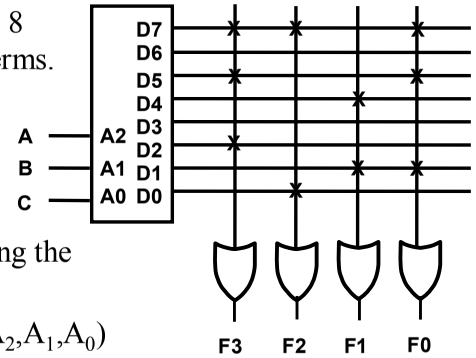
- Read Only Memories (ROM) or Programmable Read Only Memories (PROM) have:
  - N input lines,
  - M output lines, and
  - 2^N decoded minterms.
- Fixed AND array with 2^N outputs implementing all N-literal minterms.
- Programmable OR Array with M outputs lines to form up to M sum of minterm expressions.
- A program for a ROM or PROM is simply a multiple-output truth table
  - If a 1 entry, a connection is made to the corresponding minterm for the corresponding output
  - If a 0, no connection is made
- Can be viewed as a *memory* with the inputs as *addresses* of *data* (output values), hence ROM or PROM names!

Read Only Memory (cont')

- ROM components can be further classified:
  - **ROM**: programmed during manufacture and cannot be changed.
  - Programmable ROM (**PROM**): can be programmed once by the user and cannot reprogrammed afterwards.
  - Erasable PROM (**EPROM**): reprogrammable. Its initial condition can be restored by exposure to ultraviolet light, or by connecting it to high voltage (Electrically Erasable PROM, **EEPROM** or **E²PROM**).
- Writing and erasing EPROMs is much slower than reading.

Read Only Memory Example

- Example: A 8 X 4 ROM (N = 3 input lines, M= 4 output lines)
- The fixed "AND" array is a "decoder" with 3 inputs and 8 outputs implementing minterms.
- The programmable "OR" array uses a single line to represent all inputs to an OR gate. An "X" in the array corresponds to attaching the minterm to the OR

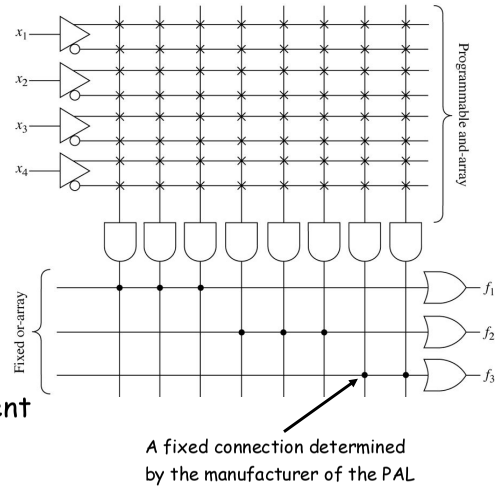


- Read Example: For input (A_2,A_1,A_0) = 011, output is (F_3,F_2,F_1,F_0) = 0011.
- What are functions F_3, F_2 , F_1 and F_0 in terms of (A_2, A_1, A_0)?

Programmable Logic Array (PAL)

- PAL has a programmable AND-array and fixed OR-array

Example:
- 4 inputs
- 3 outputs
- 8 product terms of at most 4 literals
- 3 sums:
  - 2 of 3 product terms
  - 1 of 2 product terms



- Use this PAL to implement

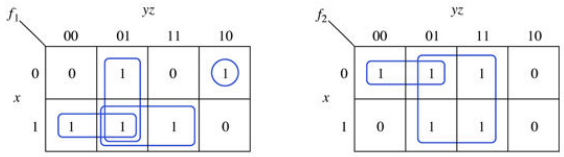
$f_1(x,y,z)=\sum_m(1,2,4,5,7)$

$f_2(x,y,z)=\sum_m(0,1,3,5,7)$

Example: Programmable Array Logic

$f_1(x,y,z)=\sum_m(1,2,4,5,7)$

$f_2(x,y,z)=\sum_m(0,1,3,5,7)$



$f_1(x,y,z) = x\bar{y} + xz + \bar{y}z + \bar{x}yz$

$f_2(x,y,z) = z + \bar{x}\bar{z}$

- Since in a straight-forward way the PAL can implement sums of at most three product terms, we write:

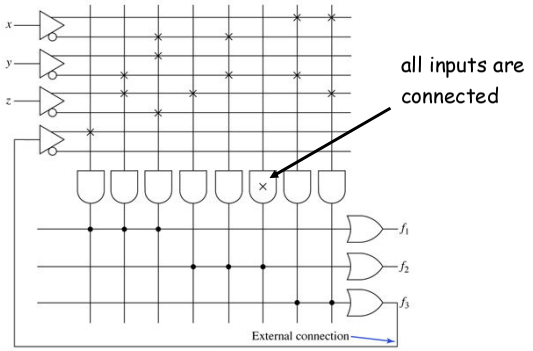
$f_1 = f_3 + \bar{y}z + \bar{x}yz$

$f_2 = z + \bar{x}\bar{z}$

$f_3 = x\bar{y} + xz$

Example: Programmable Array Logic (cont')

$$f_1 = f_3 + \bar{y}z + \bar{x}y\bar{z}$$
$$f_2 = z + \bar{x}\bar{z}$$
$$f_3 = x\bar{y} + xz$$



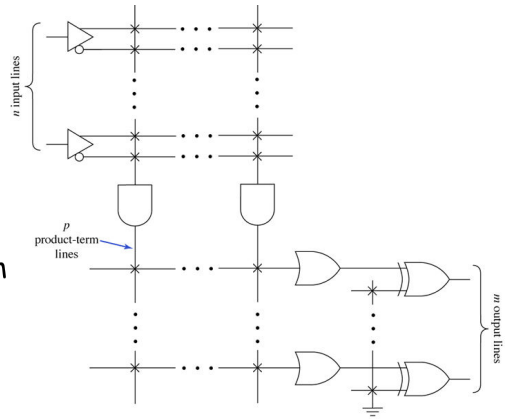
- Using a ROM would require implementing all the minterms
- PAL does not guarantee all possible Boolean functions can be implemented.

Programmable Logic Array (PLA)

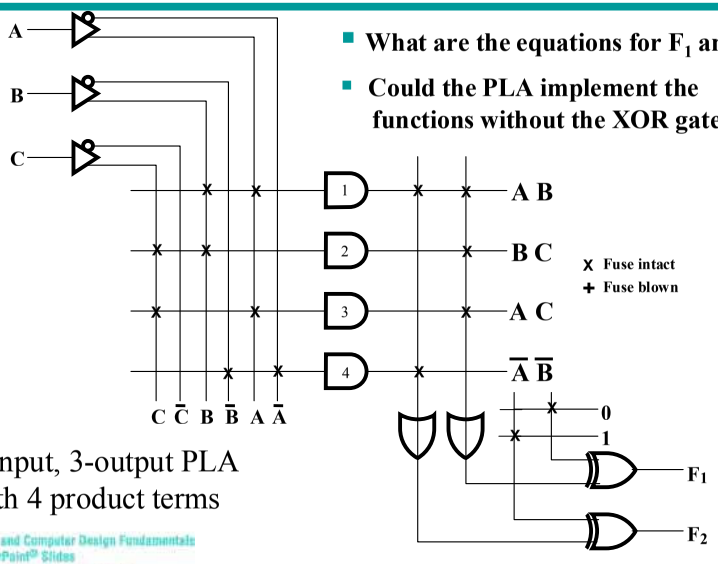
- In PLA both the AND-array and OR-array are programmable

$n \times p \times m$ PLA:

- For greater flexibility every output is available in its complemented form as well.



Programmable Logic Array Example



- **What are the equations for F_1 and F_2?**
- **Could the PLA implement the functions without the XOR gates?**
- 3-input, 3-output PLA with 4 product terms

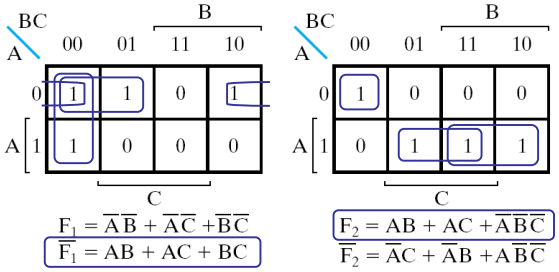
Programmable Logic Array

- The set of functions to be implemented must fit the available number of product terms
- The number of literals per term is less important in fitting
- Since output inversion is available, terms can implement either a function or its complement
- For small circuits, K-maps can be used to visualize product term sharing and use of complements
- For larger circuits, software is used to do the optimization including use of complemented functions

Example

- Implement the following functions using a 3 x 4 x 2 PLA:

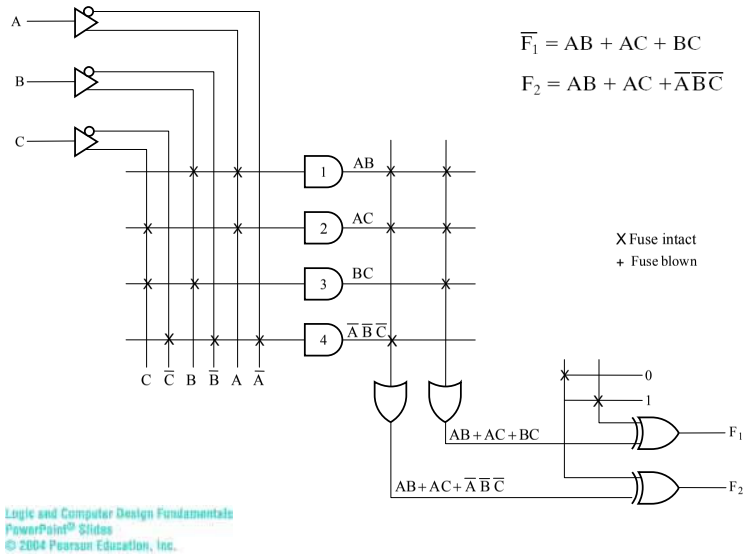
$F_1(A,B,C)=\sum_m(0,1,2,4)$ $\quad F_2(A,B,C)=\sum_m(0,5,6,7)$



- Specify the PLA connections by a PLA table:
  - "1" = connection with variable
  - "0" = connection with complement
  - "-" = no connection
  - T/C = which form of output

	Product term	Inputs A	B	C	Outputs (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$\overline{A}\,\overline{B}\,\overline{C}$	4	0	0	0	–	1

Programmable Logic Array Example

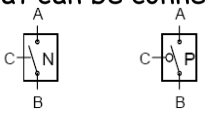


Wired-AND and Wired-OR Buses

- Implementing the medium-scale integrated (MSI) circuits we mentioned (decoders, encoders, multiplexers, LPDs) require ANDing and ORing of many signals.
- AND/OR gates with large fan-in consume more power, are more expensive, and might not be available.
- Solutions:
  - "Tree" of gates.
  - Wired-AND and Wired-OR buses.

Implementing Logic Gates using Switches

- The technological implementation of logic gates is done using transistors that implement switches.
- A switch is connected to 3 wires:
  - A control signal (C)
  - 2 wire endpoints that can be connected by the switch (A,B)



- We define two types of switches:

P switch

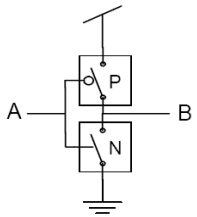
C	Switch status
0	connected
1	disconnected

N switch

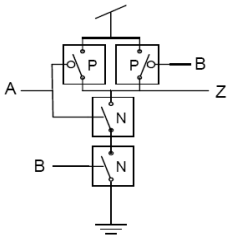
C	Switch status
0	disconnected
1	connected

Implementing Logic Gates using Switches (cont')

- Examples
  1. Building an inverter
     $B=\bar{A}$

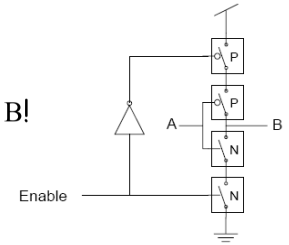


  2. Building a NAND gate
     Z=NAND(A,B)

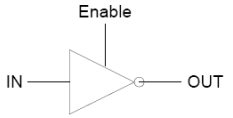


3-State Inverter

- We define a new gate using switches:
  - Enable=1: Inverter
  - Enable=0: No logic value assigned to B! In this case we say the value of B is Hi-Z.

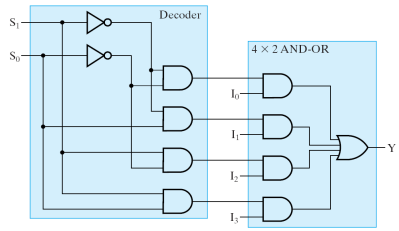


The gate is called 3-state inverter

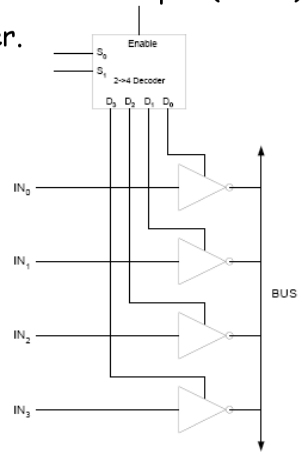


3-State Bus

- It is possible to connected several outputs of 3-state gates, as long as only one of them writes to the common output (=BUS).
- To guarantee this we can use a decoder.
- This component (BUS+decoder) is actually a 4-to-1 multiplexer

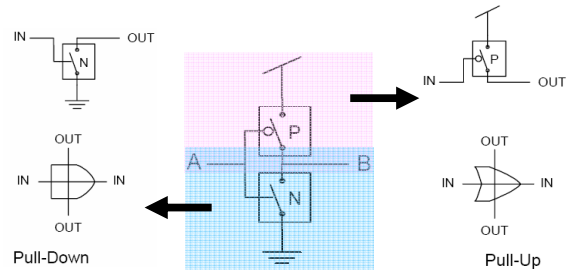


4-to-1 multiplexer



Pull-Down and Pull-Up Gates

- To remove the need for a decoder we define 2 new gates based on the NOT gate:

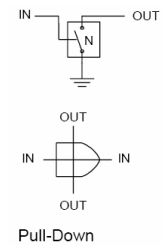


IN=1 ➔ OUT=0
IN=0 ➔ OUT=Hi-Z

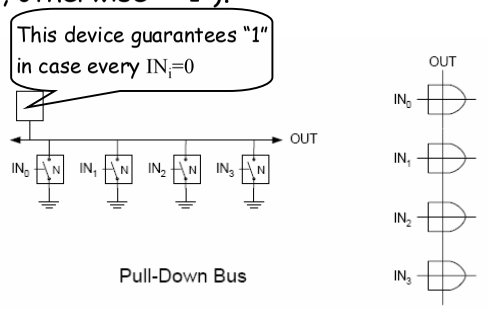
IN=0 ➔ OUT=1
IN=1 ➔ OUT=Hi-Z

Wired-AND BUS

- Connecting several pull-down gates to a common output results in a **Wired-AND Bus** (if at least one output is "1" then the value on the bus is "0", otherwise – "1").



Pull-Down



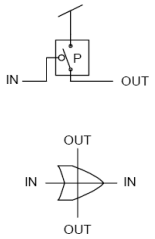
Pull-Down Bus

IN=1 ➔ OUT=0
IN=0 ➔ OUT=Hi-Z

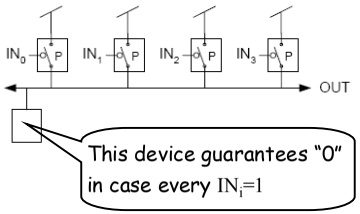
- The implemented function is NOR (or AND of the complements) $NOR(IN_0, IN_1, ..., IN_k) = AND(\overline{IN_0}, \overline{IN_1}, ..., \overline{IN_k})$

Wired-OR BUS

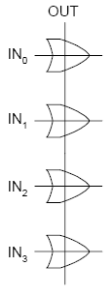
- Connecting several pull-up gates to a common output results in a **Wired-OR Bus** (if at least one output is "0" then the value on the bus is "1", otherwise – "0").



Pull-Up



Pull-Up Bus



IN=0 ➔ OUT=1
IN=1 ➔ OUT=Hi-Z

- The implemented function is NAND (or OR of the complements) $NAND(IN_0, IN_1, ..., IN_k) = OR(\overline{IN_0}, \overline{IN_1}, ..., \overline{IN_k})$