www.covenantuniversity.edu.ng

Covenant University

Raising a new Generation of Leaders

CEN510:
Digital Systems Design with VHDL

Course Lecturer: Dr. Joke Badejo

& Engr. Temitope Takpor

Department of Electrical & Information Engineering

REFERENCE TEXT

- Ronald J. Tocci, Neal S. Widmer and Gregory L. Moss, ***Digital Systems: Principles and Applications***, Pearson Education, Upper Saddle River, NJ 07458. *Any later edition from 10th(2007)*
- Stephen Brown and Zvonko Vranesic, ***Fundamentals of Digital Logic with VHDL Design***. *Any later edition from 3rd(2009)*
- Pong P. Chu, ***RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability***, Wiley-Interscience. *Any later edition from 1st (2006)*
- Floyd, **Digital Fundamentals**, Pearson Education, Upper Saddle River, NJ 07458, 2009. *Any later edition from 10th(2009)*
- Kris Gaj, **ECE 545 - Digital System Design with VHDL Lecture Notes**, George Mason University

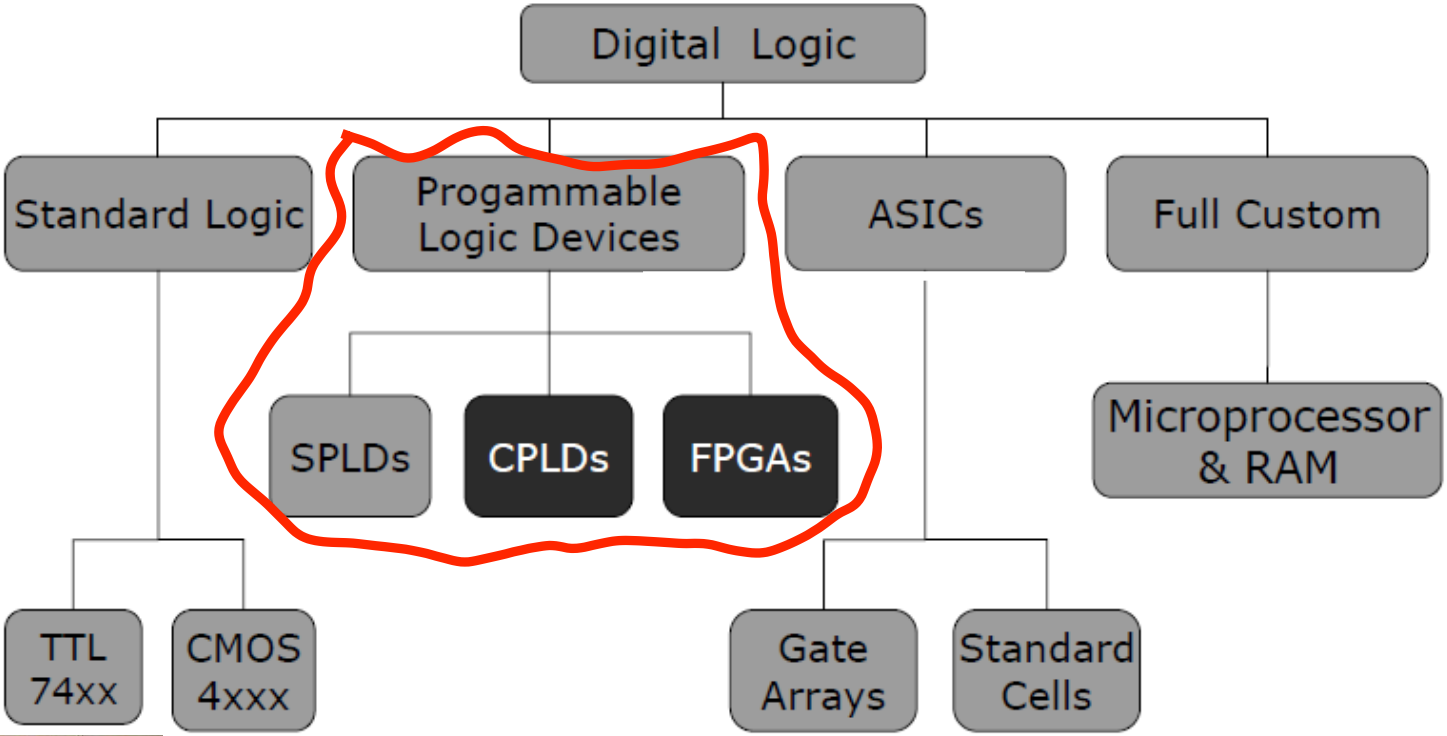
Module 2

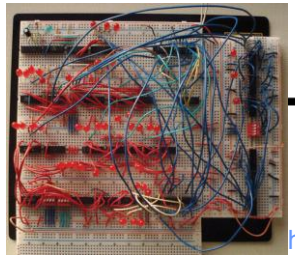
- ***Reconfigurable*** **Logic Devices**
  - **Overview of digital logic devices**
  - **Classification of device technologies**
  - **Programmable Logic Devices (PLD)**
  - **FPGA Architecture**
  - **CPLD vs. FPGA**

Digital System (Logic) Design

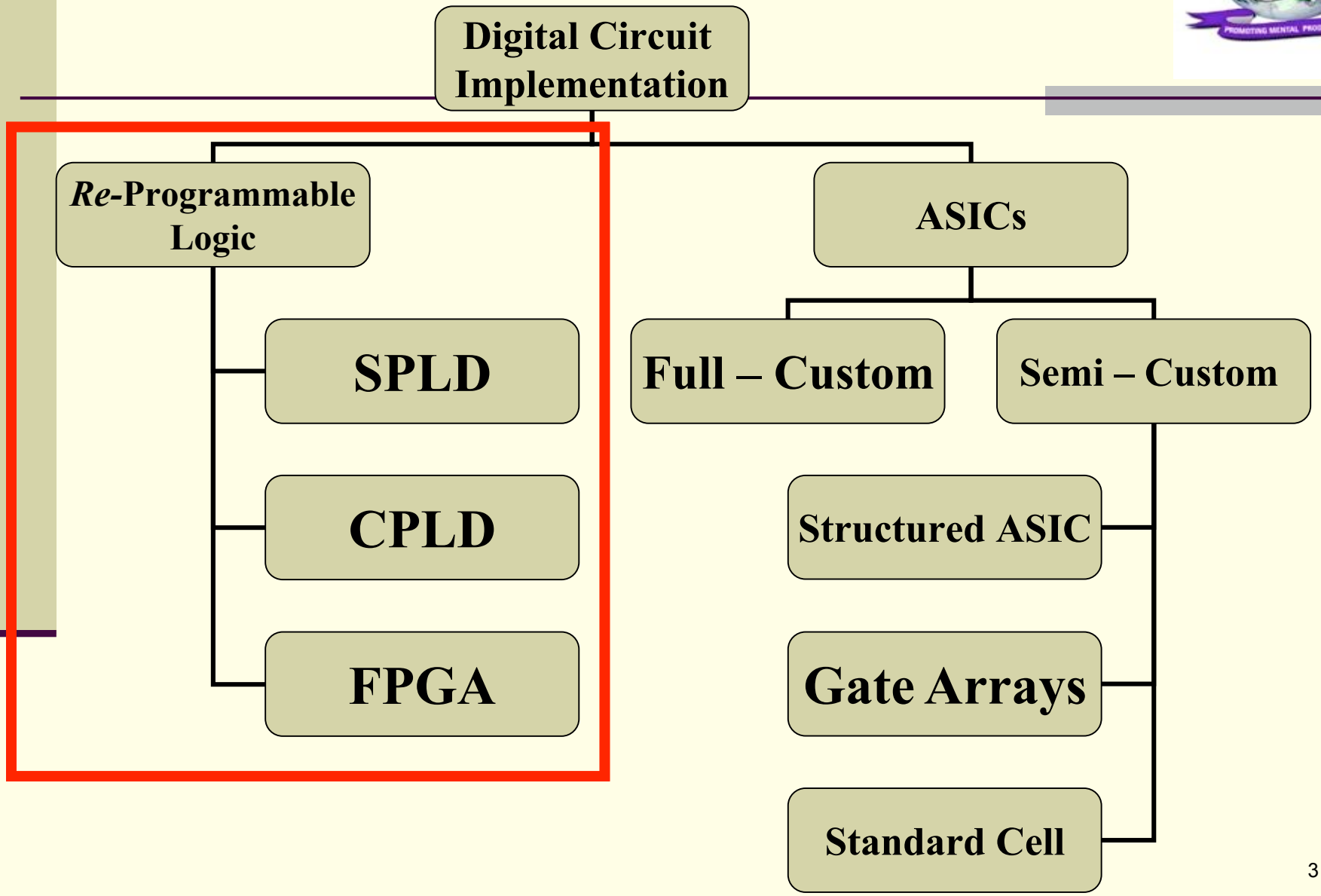
- Digital circuit or logic is basically made up of ICs.
- Logic circuits are designed based on logic (decisions).
- Logic design tools to be adopted depends largely on device technology
  - Boolean algebraic expression
  - Truth tables
  - Schematic capture
  - Timing diagrams
  - Logic behavioral description with languages (Verilog or VHDL)

Implementation Technologies





Digital Logic Technologies



Programmable Logic Device (PLD)

- A PLD is an integrated circuit with internal logic gates and interconnects.
- These gates can be connected to obtain the required logic configuration.
- The term "programmable" means changing either hardware or software configuration of an internal logic and interconnects.
- The configuration of the internal logic is done by the user.
- With PLDs, the focus shifted from architecture of an actual device to hardware description methods via hardware description languages (HDL).
- PROM, EPROM, PAL, GAL, ASIC, SPLD, CPLD and FPGA are examples of Programmable Logic Devices.

Why Programmable Chips?

- As compared to hard-wired chips, programmable chips can be customized, according to user needs, by programming or reconfiguring.
- This *convenience*, coupled with the option of *re-programming* in case of errors, makes the programmable chips very attractive.
- Other benefits include *instant turnaround*, *low starting cost* and *low risk*.

Why Programmable Chips?

- As compared to reprogrammable chips, ASIC (Application Specific Integrated Circuit) has a longer design cycle and were more costly.
- Still, ASIC has its own market due to the added benefit of faster performance and lower cost if produced in high volume.
- Programmable chips are good for medium to low volume products. If you need more than 10,000 chips, go for ASIC or hard copy.

Classification of device technologies

- Based on where customization (or reconfiguration) is done:
  - In a fab (fabrication facility): ASIC (Application Specific IC)
    - Full-custom ASIC
    - Standard cell ASIC
    - Gate array ASIC
  - In the "field": non-ASIC, reprogrammable
    - Off-the-shelf SSI (Small Scaled IC)/MSI
    - Simple programmable logic device (SPLD)
    - Complex programmable logic device (CPLD)
    - Field programmable gate array (FPGA)

Full-custom ASIC

- All aspects (e.g., size of a transistor) of a circuit are tailored for a particular application.
- Circuit fully optimized
- Design extremely complex and involved
- Only feasible for small components
- Masks needed for all layers

Standard-Cell ASIC

- Circuit made of a set of pre-defined logic, known as standard cells
- E.g., basic logic gates, 1-bit adder, D FF etc
- Layout of a cell is pre-determined, but layout of the complete circuit is customized
- Masks needed for all layers

Gate array ASIC

- Circuit is built from an array of a single type of cell (known as base cell)
- Base cells are pre-arranged and placed in fixed positions, aligned as one- or two-dimensional array
- More sophisticated components (macro cells) can be constructed from base cells
- Masks needed only for metal layers (connection wires)

SSI/MSI components

- Small parts with fixed, limited functionality
- E.g., 7400 TTL series (more than 100 parts)
- Resource (e.g., power, board area, manufacturing cost etc.) is consumed by "package" but not "silicon"
- No longer a viable option

Simple Field Programmable Device

- Programmable device with simple internal structure
  - PROM (Programmable Read Only Memory)
  - PAL (Programmable Array Logic)
  - GAL (Generic Array Logic)
- No custom mask needed
- Replaced by CPLD/FPGA

Complex Field Programmable Device

- Device consists of an array of generic logic cells and general interconnect structure
- Logic cells and interconnect can be "programmed" by utilizing "semiconductor fuses or "switches" .
- Customization is done "in the field" .
- Two categories:
  - CPLD (Complex Programmable Logic Device)
  - FPGA (Field Programmable Gate Array)
- No custom mask needed.

Summary

Programmable Logic

Programmable Logic Devices (PLDs) are ICs with a large number of gates and flip flops that can be configured with basic software to perform a specific logic function or perform the logic for a complex circuit. Major types of PLDs are:

SPLD: (Simple PLDs) are the earliest type of array logic used for fixed functions and smaller circuits with a limited number of gates. (The PAL and GAL are both SPLDs).

CPLD: (Complex PLDs) are multiple SPLDs arrays and inter-connection arrays on a single chip.

FPLD: (Field Programmable Gate Array) are a more flexible arrangement than CPLDs, with much larger capacity.

Summary

Programmable Logic

Advantages to PLDs include

- Reduced complexity of circuit boards
  - Lower power requirements
  - Less board space
  - Simpler testing procedures
- Higher reliability
- Design flexibility

Digital System design with PLDs

- a detailed view

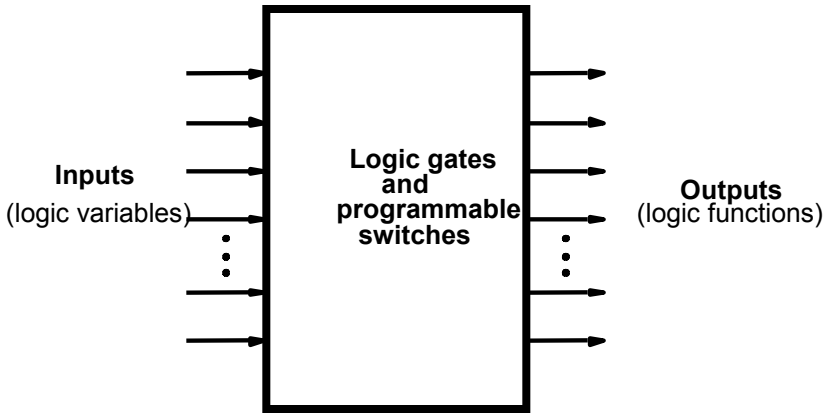
Programmable Logic Devices

- Standard TTL and CMOS ICs have a limited number of gates and this number is fixed
- Inefficient for developing large circuits
  - Take up too much space on the board
- It is possible to manufacture ICs with a large number of gates with a structure which is **NOT** fixed
  - Introduced in the 1970s
  - PLDs




Programmable Logic Devices

- A PLD is a general purpose IC:
  - A collection of logic circuit elements
  - Can be customized in different ways
  - Can be re-used
  - Can be viewed as “Black Box” with logic gates and programmable switches (connections)



Programmable Logic Devices

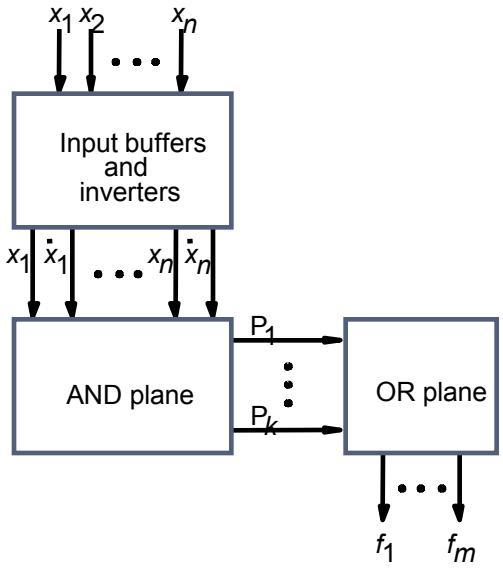
- PLA (programmable logic array)
  - – The first PLD on the market
  - – Two level AND/OR array structure with user programmable connections
  - – Programmable AND array, programmable OR array
- PAL (programmable array logic)
  - – Appeared on the scene after PLA s
  - – Lower cost
  - – The MSI of the programmable industry; sometimes simply called PLD
  - – Advanced to GAL (generic array logic)
  - – Programmable AND array, fixed OR array

Programmable Logic Devices

- CPLD (complex programmable logic device)
  - – A collection of PLD s on a chip with programmable on-chip interconnections
  - – Contains macrocells
- FPGA (field programmable gate array)
  - – Large number of basic logic blocks (simple gates) with programmable
  - interconnection
  - – Consists of an array of logic cells
  - – Each logic cell contains a flipflop (FF), lookup table (LUT) and supporting logic such as multiplexers, carry terms for adders etc.

Programmable Logic Array

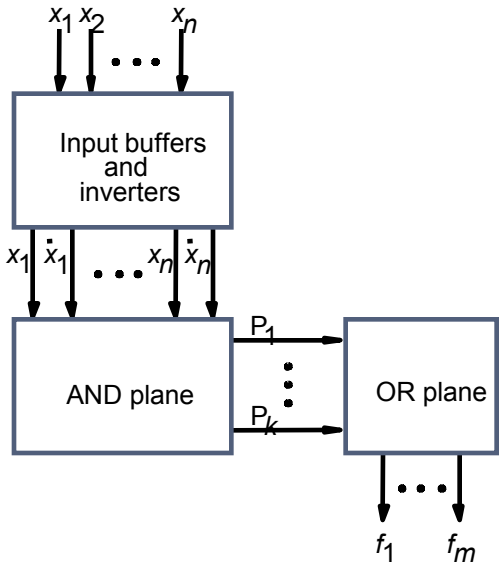
- Programmable Logic Array (PLA):
  - First to be developed
  - Architecture:



Programmable Logic Array

- Programmable Logic Array (PLA):
  - First to be developed
  - Architecture:

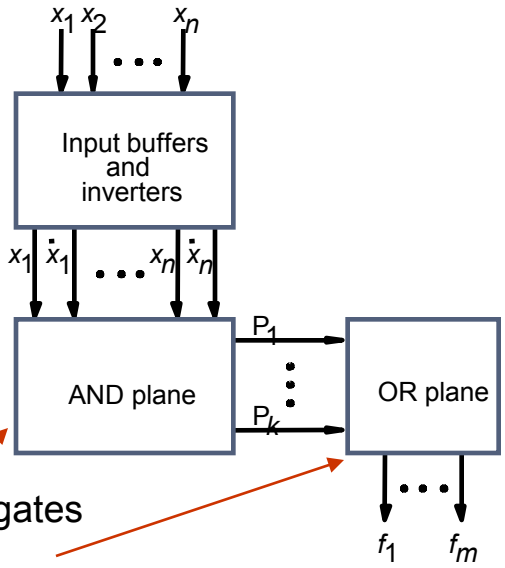
Based on the fact that logic functions can be realized in sum-of-product form.



Programmable Logic Array

- Programmable Logic Array (PLA):
  - First to be developed
  - Architecture:

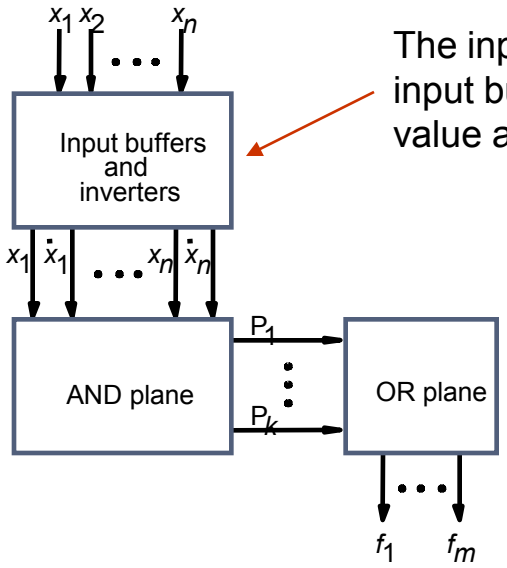
Based on the fact that logic functions can be realized in sum-of-product form.



Comprised of a collection of AND gates whose outputs are connected to a collection of OR gates

Programmable Logic Array

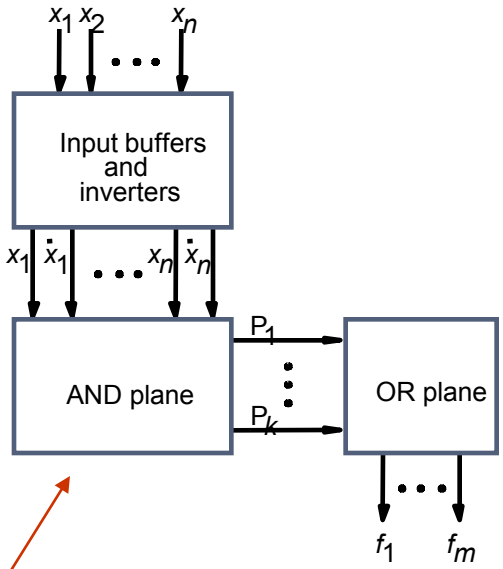
- Programmable Logic Array (PLA):
  - First to be developed
  - Architecture:



The inputs are fed through a series of input buffers that provide the the true value and the compliment of each input

Programmable Logic Array

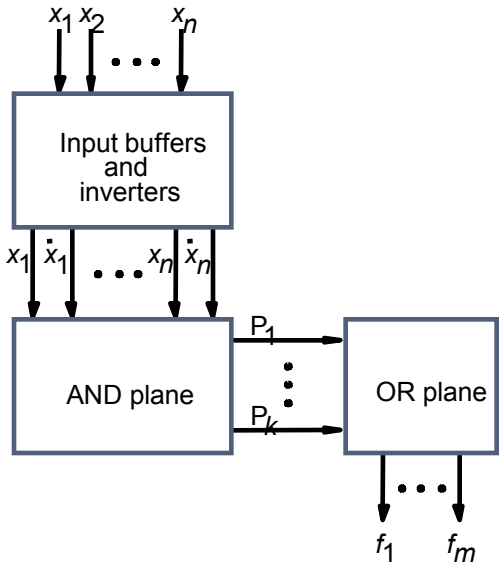
- Programmable Logic Array (PLA):
  - First to be developed
  - Architecture:



The AND plane produces a set of product terms

Programmable Logic Array

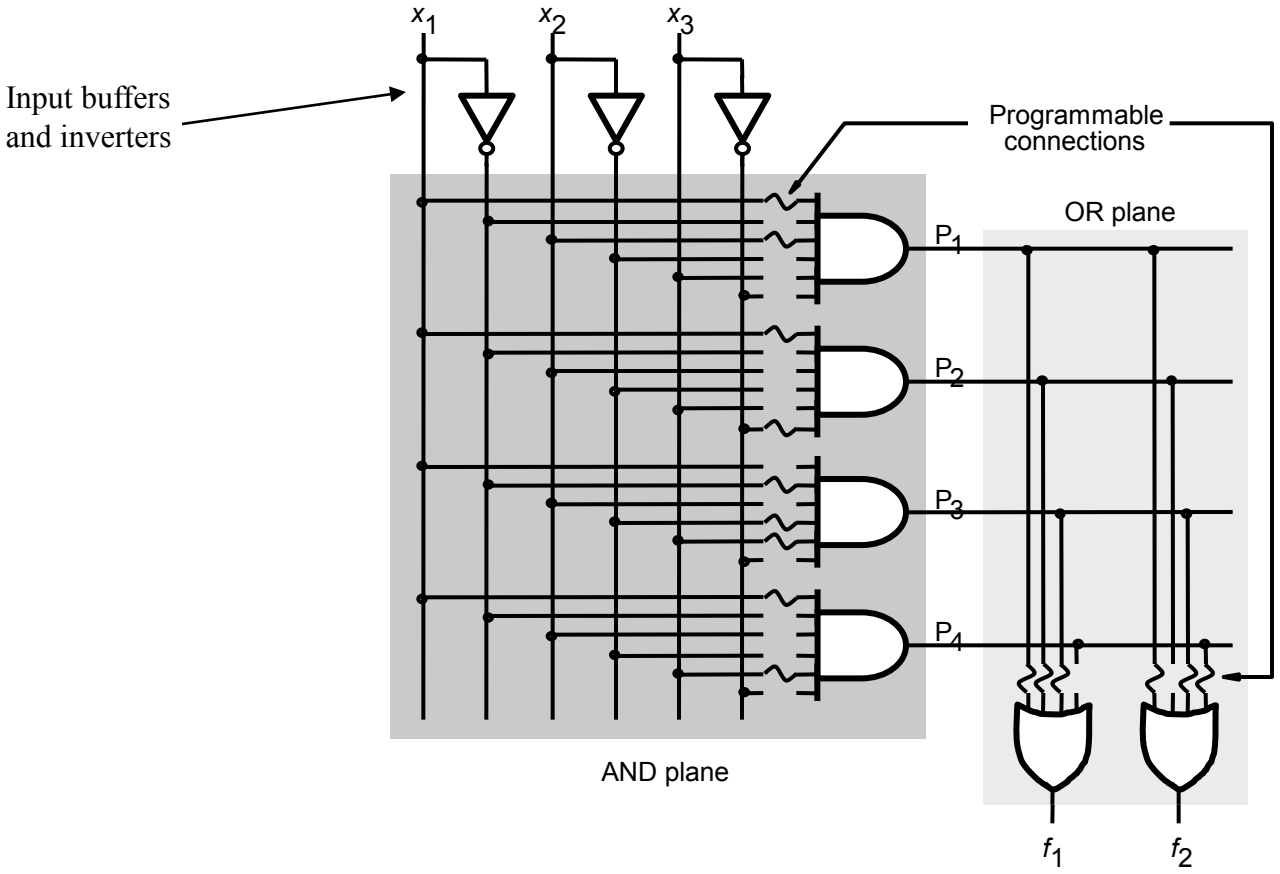
- Programmable Logic Array (PLA):
  - First to be developed
  - Architecture:



The OR plane produces the summing terms

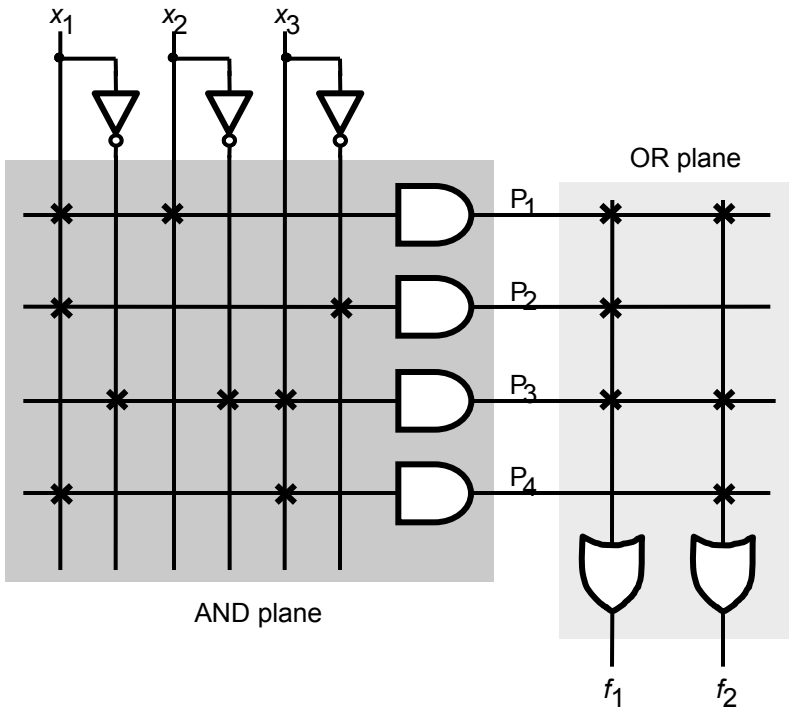
Programmable Logic Array

- More detailed diagram:



Programmable Logic Array

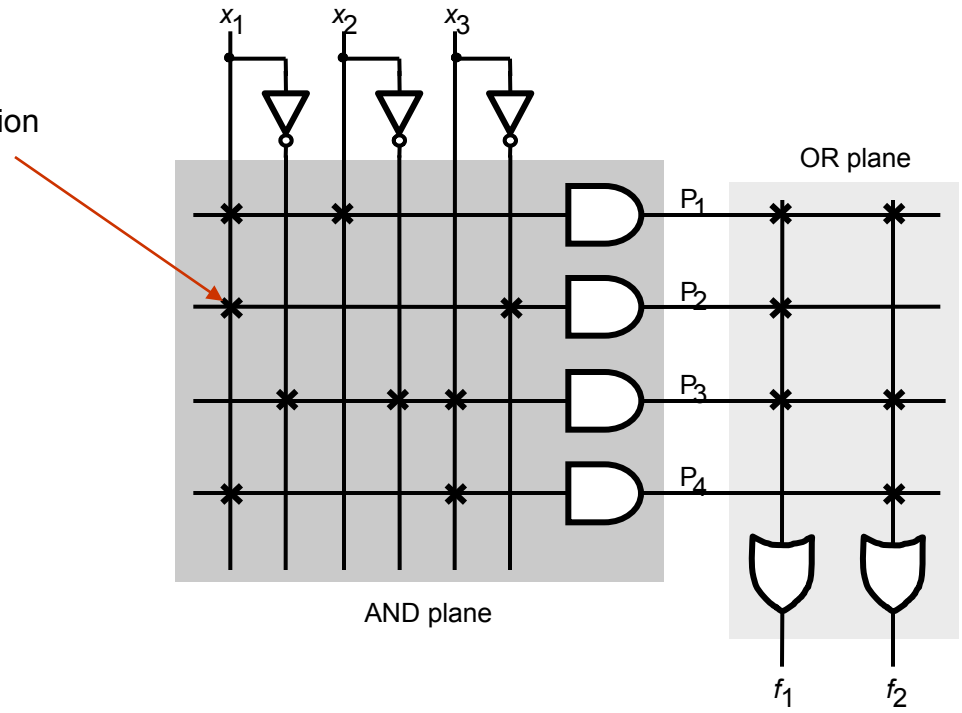
- Typical schematic diagram:



Programmable Logic Array

- Typical schematic diagram:

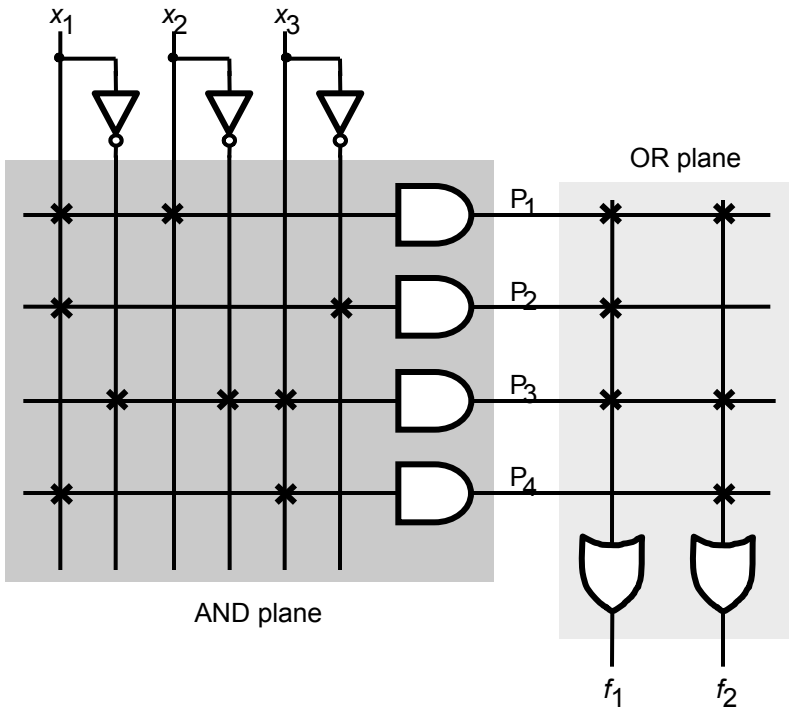
The Xs indicate a programmed connection



What is the expression for P_1?

Programmable Logic Array

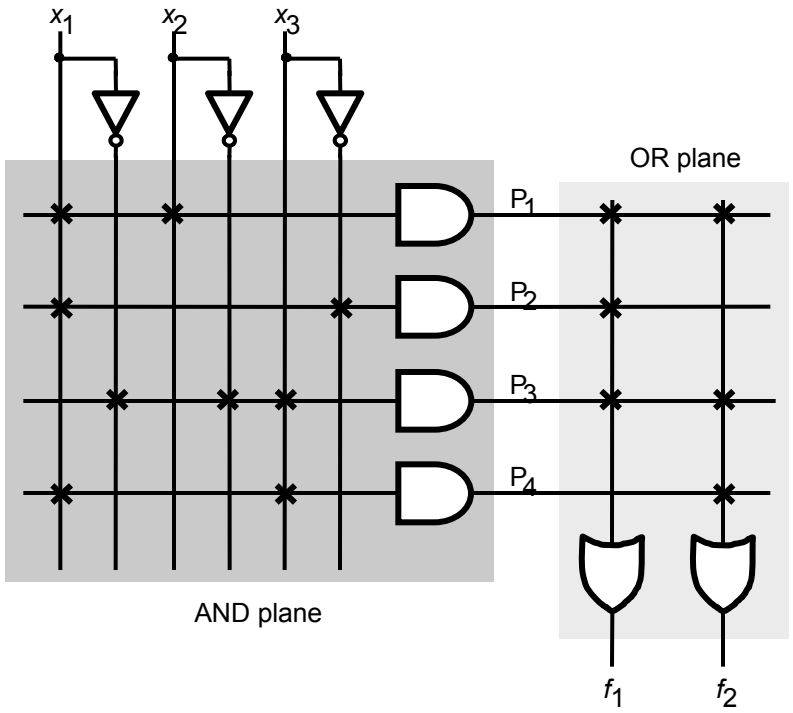
- Typical schematic diagram:



What is the expression for $P_1 = X_1$ AND X_2

Programmable Logic Array

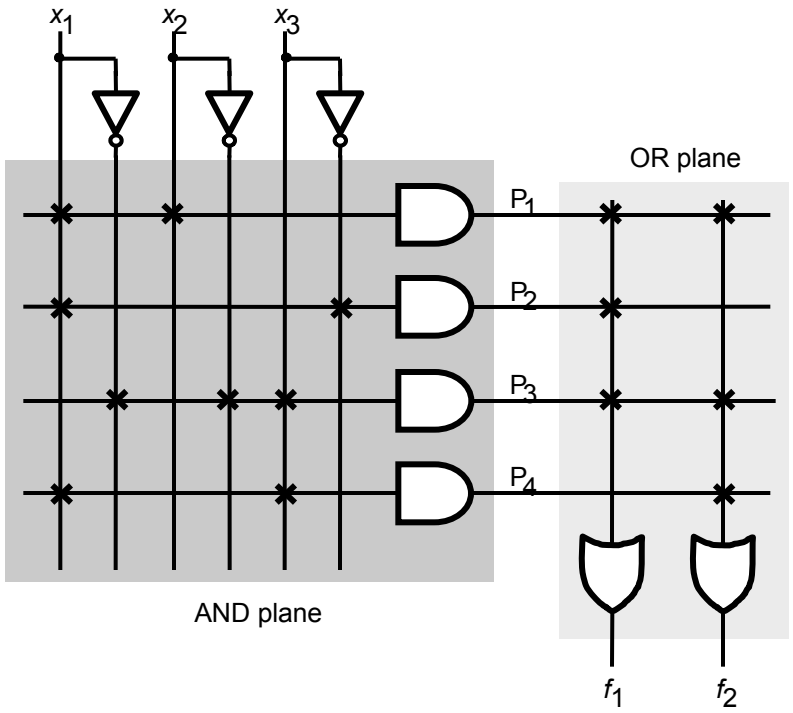
- Typical schematic diagram:



What is the expression for P_2 ?

Programmable Logic Array

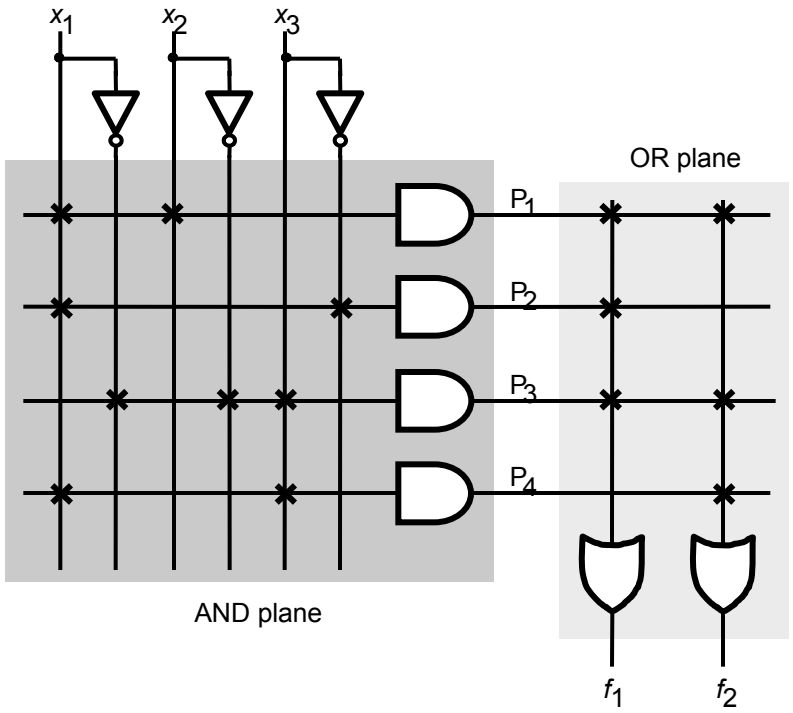
- Typical schematic diagram:



What is the expression for $P_2 = X_1$ AND NOT X_3

Programmable Logic Array

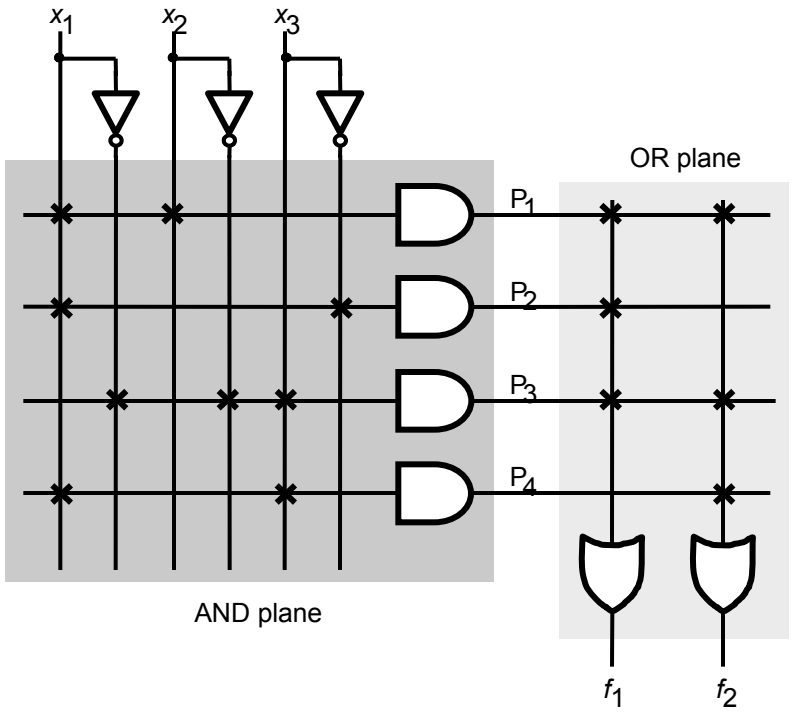
- Typical schematic diagram:



What is the expression for f_1 ?

Programmable Logic Array

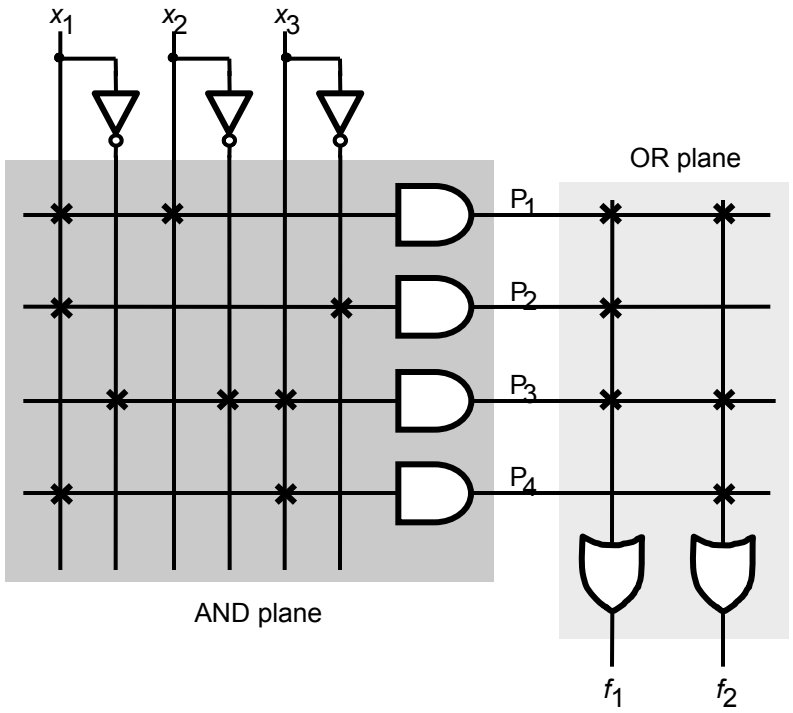
- Typical schematic diagram:



What is the expression for f_1 = (X_1 AND X_2) + (X_1 AND NOT X_3) + (NOT X_1 AND NOT X_2 AND X_3)

Programmable Logic Array

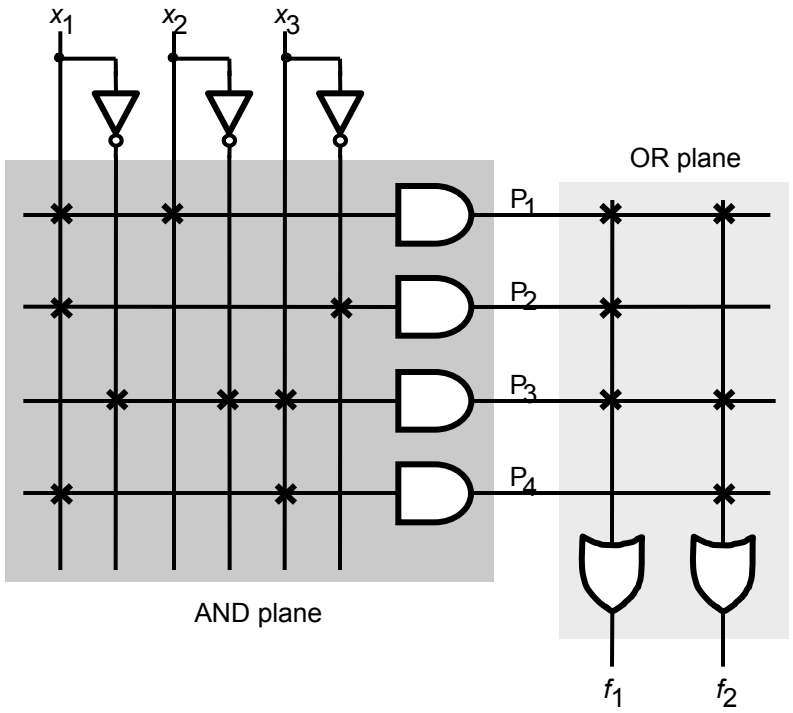
- Typical schematic diagram:



What is the expression for f_2 ?

Programmable Logic Array

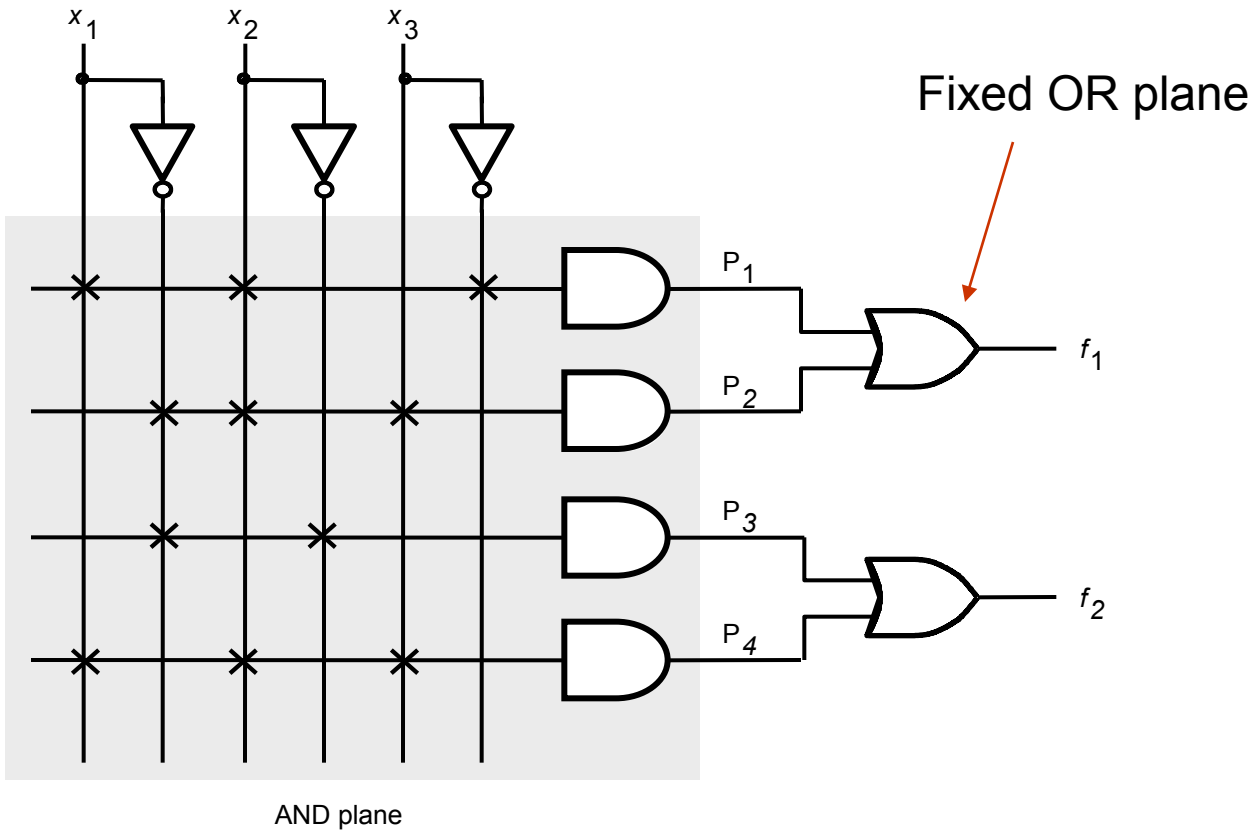
- Typical schematic diagram:



What is the expression for f_2 = (X_1 AND X_2) + (NOT X_1 AND NOT X_2 AND X_3) + (X_1 AND X_3)

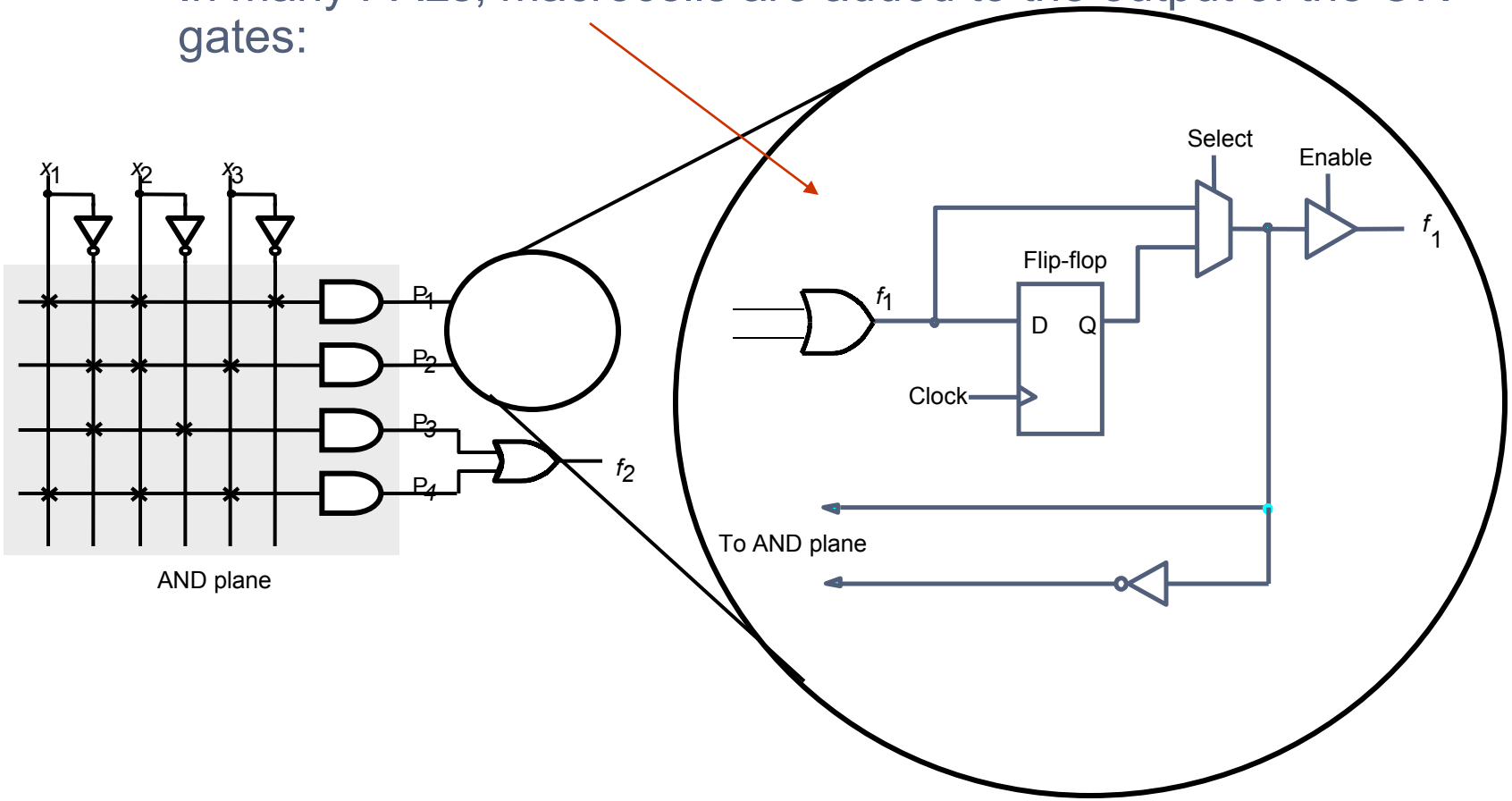
Programmable Array Logic

- Typical schematic diagram of a PAL:



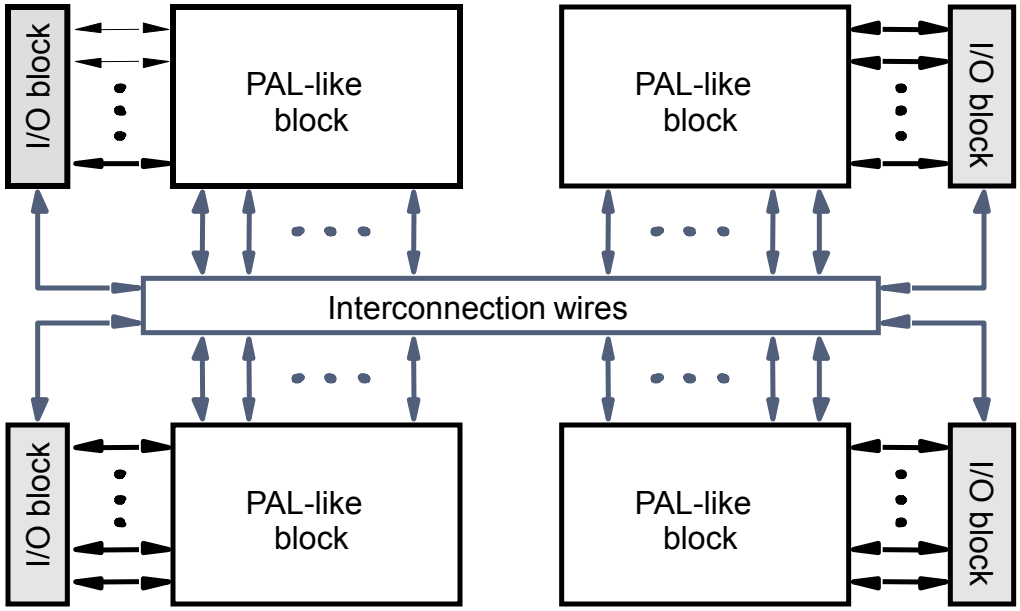
Programmable Array Logic

- In many PALs, macrocells are added to the output of the OR gates:



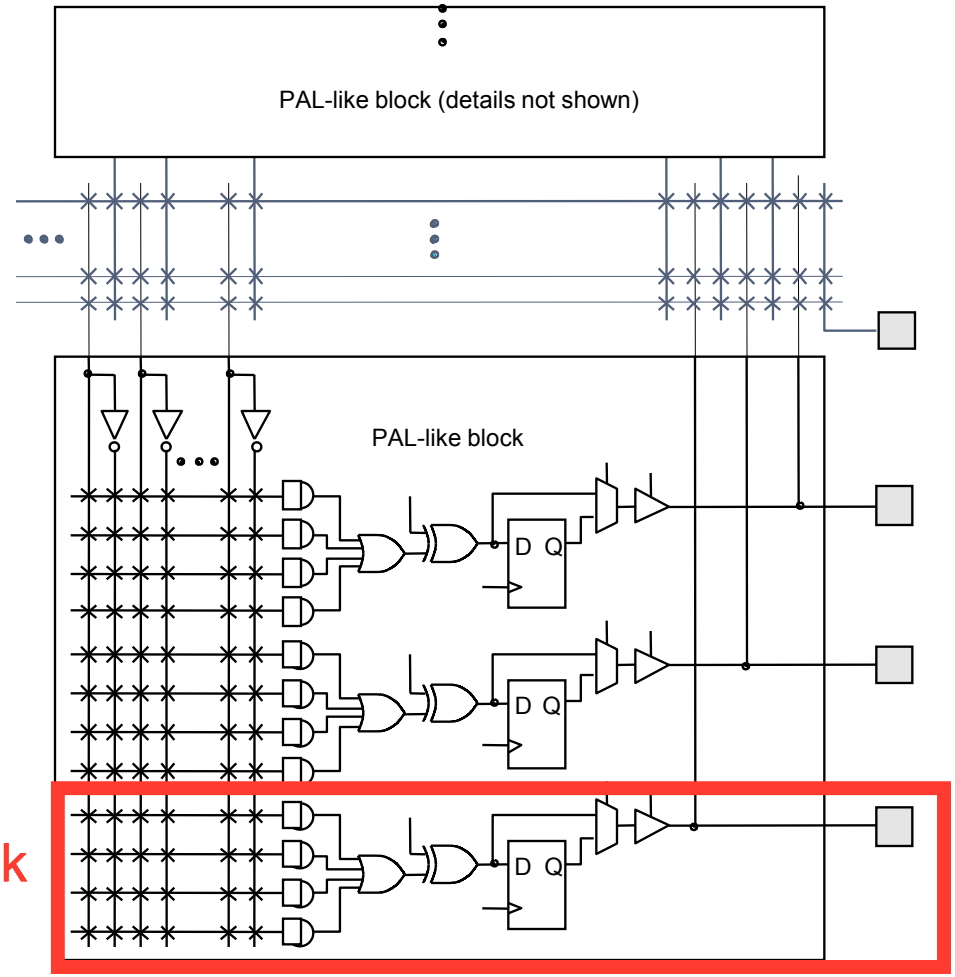
Complex Programmable Logic Devices

- CPLDs are made up of many PAL-like devices with programmable switches



Complex Programmable Logic Devices

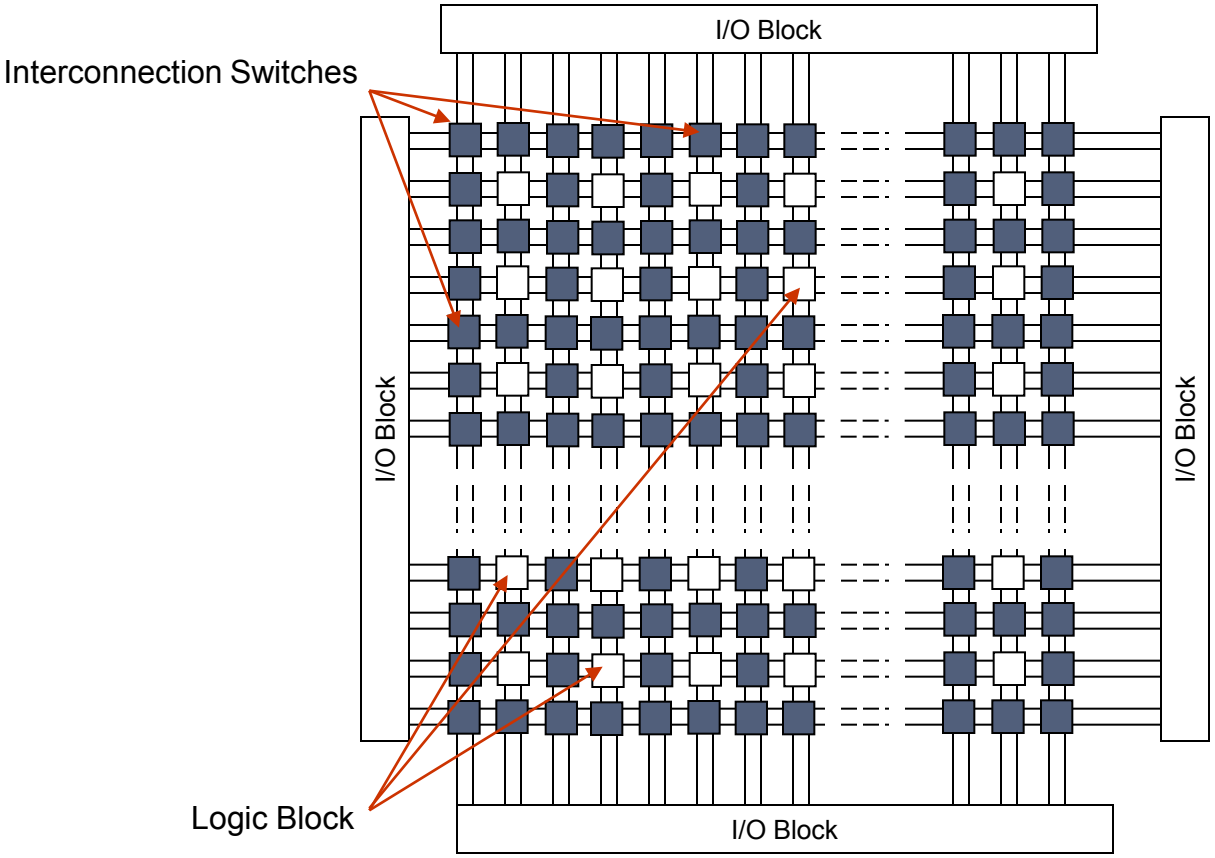
- A closer look



A PAL-like block

Field Programmable Gate Array

- General structure of an FPGA:



Field Programmable Gate Array

- Section of a programmed FPGA:

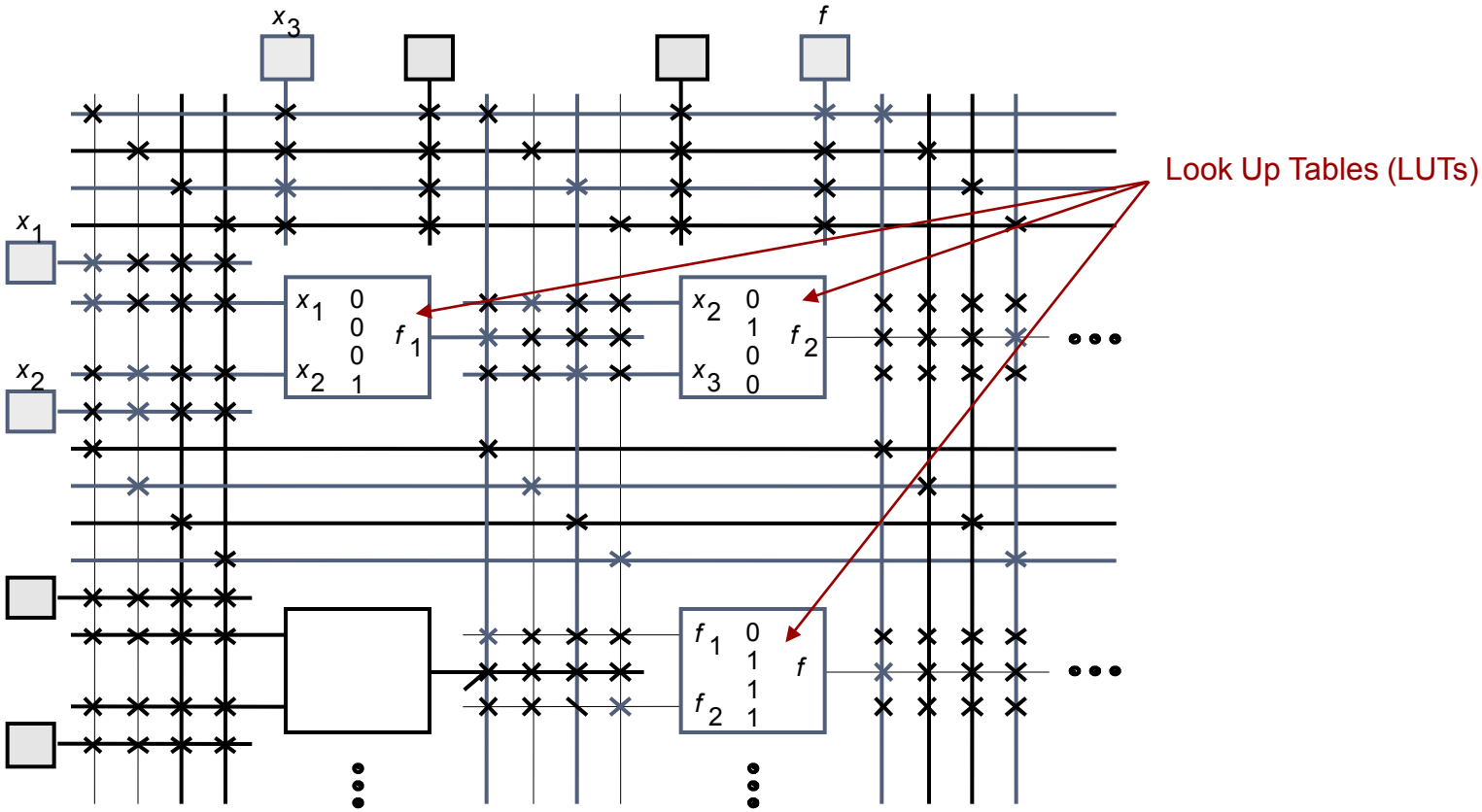
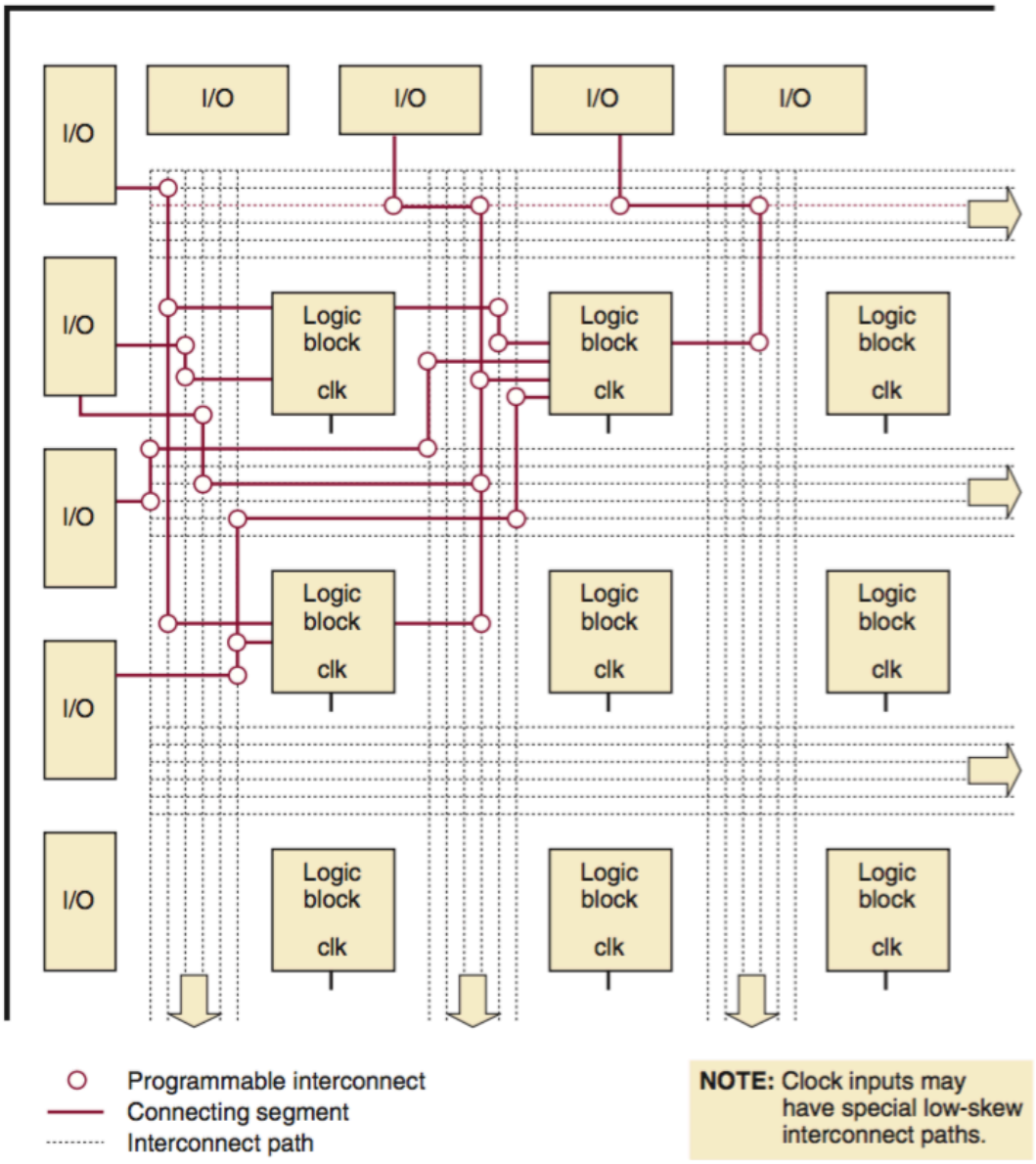
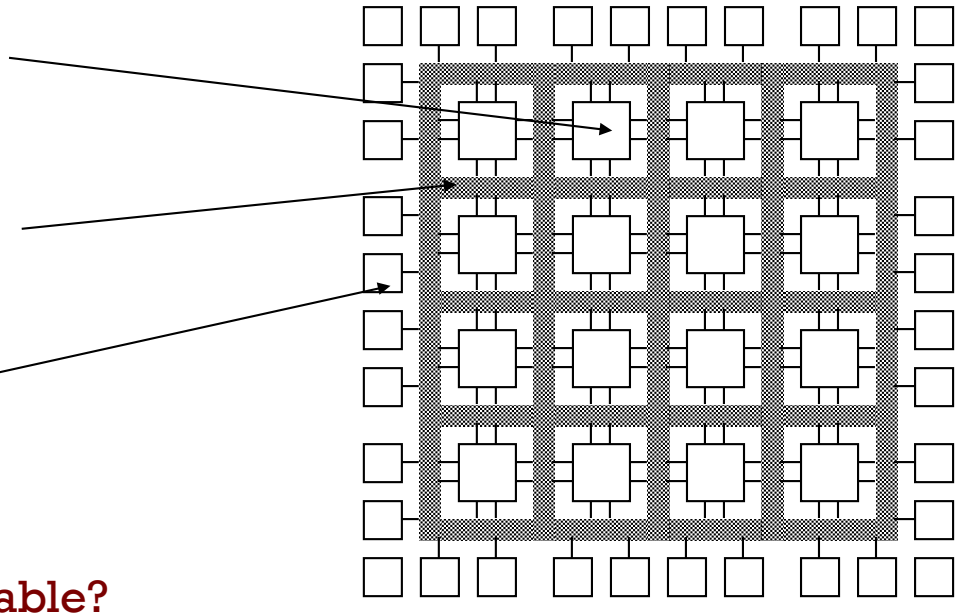


FIGURE 13-2 FPGA architecture.



Field-Programmable Gate Arrays structure

- Logic blocks
  - To implement combinational and sequential logic
- Interconnect
  - Wires to connect inputs and outputs to logic blocks
- I/O blocks
  - Special logic blocks at periphery of device for external connections
- Key questions:
  - How to make logic blocks programmable?
  - How to connect the wires?



FPGA Structure

- Typical organization in 2-D array
  - Configurable logic blocks (CLBs) contain functional logic (could be similar to PAL22V10)
    - Combinational functions plus FFs
    - Complexity varies by device
  - CLB interconnect is either local or long line
    - CLBs have connections to local neighbors
    - Horizontal and vertical channels use for long distance
    - Channel intersections have switch matrix
  - IOBs (I/O logic Blocks) connect to pins
    - Usually have some additional C.L./FF in block

Field Programmable Gate Arrays (FPGAs)

- FPGAs have much more logic than CPLDs
  - 2K to >10M equivalent gates
  - Requires different architecture
  - FPGAs can be RAM-based or Flash-based
    - RAM FPGAs must be programmed at power-on
      - External memory needed for programming data
      - May be dynamically reconfigured
    - Flash FPGAs store program data in non-volatile memory
      - Reprogramming is more difficult
      - Holds configuration when power is off

PLDs

- PAL
  - Typically has about 8 macrocells
  - Can implement circuits of up to 160 logic gates
- CPLD
  - Typically has about 500 macrocells
  - Can implement circuits of up to 10,000 logic gates
- FPGA
  - Can implement larger circuits than CPLDs
  - Do not contain AND or OR planes
  - Made up of logic blocks




PLDs

- It will be important to understand the structure of the PLD you will use
  - Need one with enough gates and circuits to implement the design
  - You don't want to pay for a PLD that is larger than necessary
- Most VHDL ISEs can synthesize the design into a logic circuit



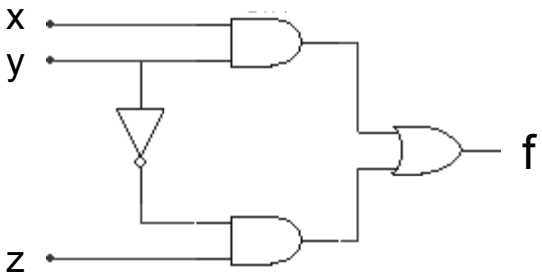

VHDL

- VHDL is a two-level acronym
  - HDL: Hardware Description Language
  - V: an acronym for Very High Speed Integrated Circuit
- Developed in the 1980's
  - DoD



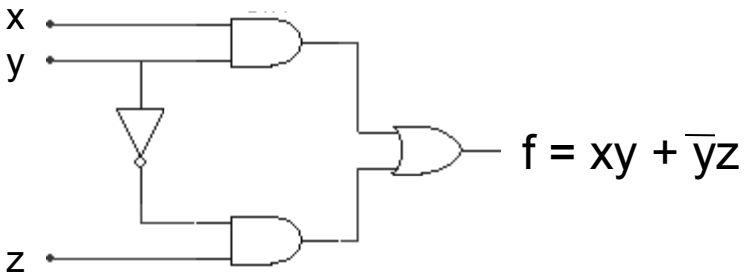

VHDL Example #1

- A simple example



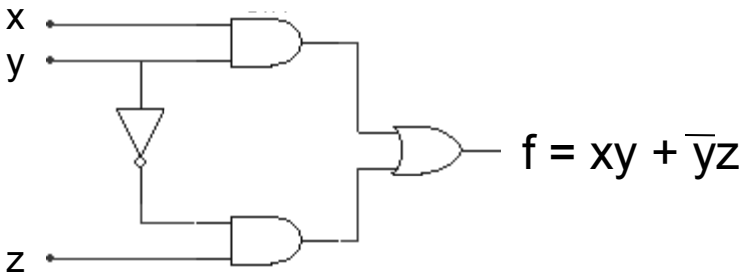
VHDL Example #1

- A simple example



VHDL Example #1

- A simple example



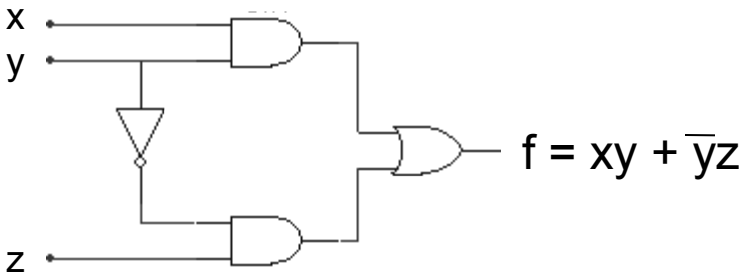
Below is the VHDL code for the circuit above.

```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                        f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```




VHDL Code Explanation

- A simple example



Below is the VHDL code for the circuit above. It is composed of 2 parts:

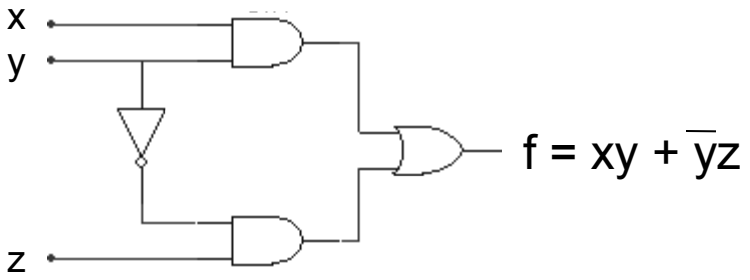
```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                      f     : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct

The ARCHITECTURE construct

VHDL Code Explanation: Entity

- A simple example

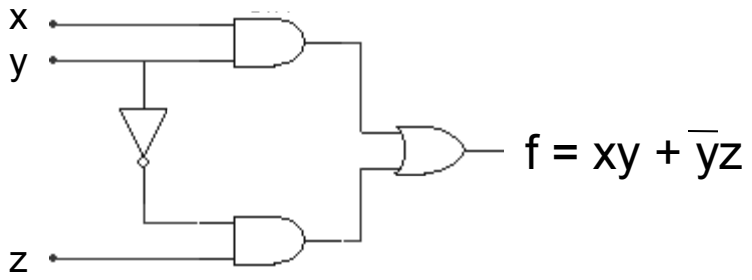



```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                      f     : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

VHDL Code Explanation: Entity

- A simple example



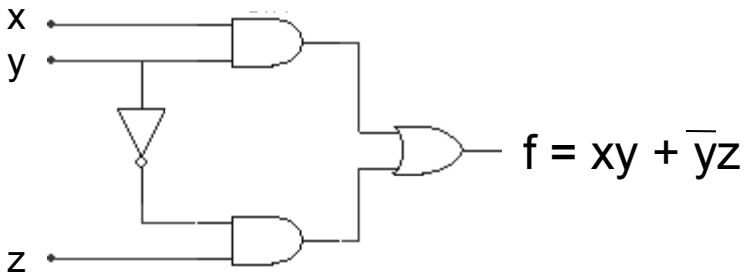

```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                    f       : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

The entity must be named:
in this case “example1”

VHDL Code Explanation: Entity

- A simple example




```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                    f       : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

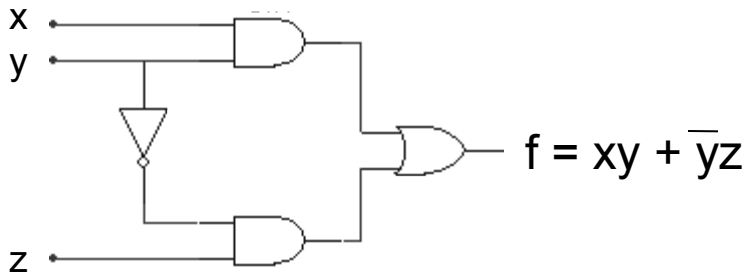
The ENTITY construct is analogous to a symbol in a schematic diagram

The input and output signals are called “ports” and must be identified by the ***keyword*** **PORT**

A *port* is an input or output connection: a single signal

VHDL Code Explanation: Entity

- A simple example



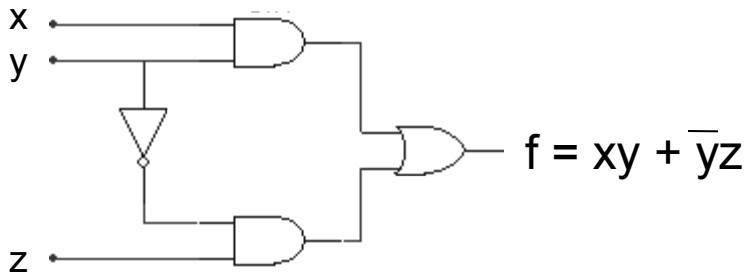

```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                            f      : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

Each port (there are four in this entity) has an associated mode: **IN** or **OUT**

VHDL Code Explanation: Entity

- A simple example



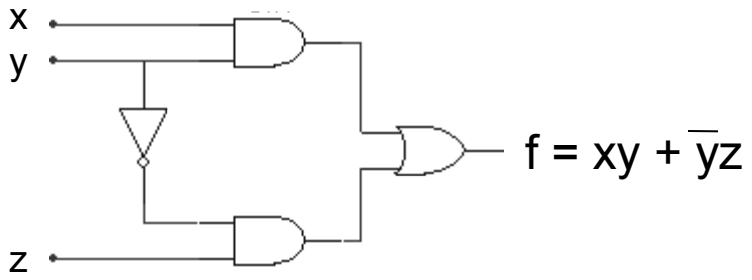

```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                      f     : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

Each port also has an associated type: in this case they are all of type **BIT**

VHDL Code Explanation: Entity

- A simple example



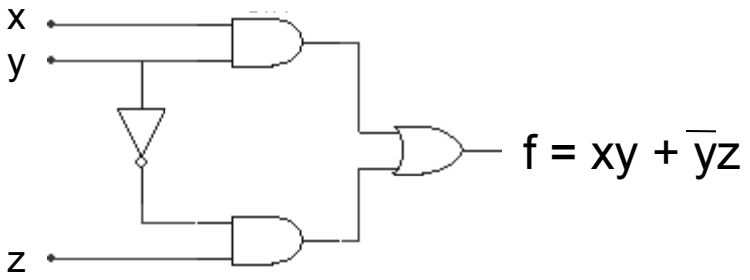

```
ENTITY example1 IS
        PORT ( x, y, z      : IN   BIT ;
                    f       : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

An entity has a beginning and an end.
The beginning is marked by the keyword: **ENTITY**
The end is marked by the keyword **END**

VHDL Code Explanation: Entity

- A simple example



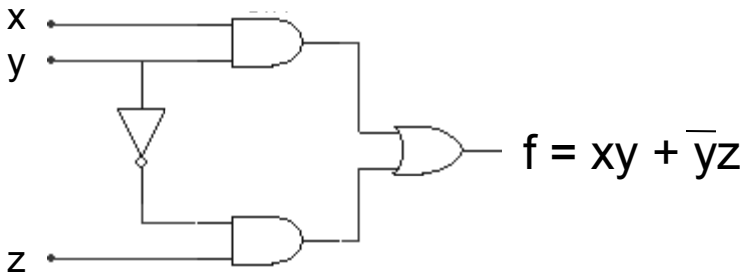

```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                    f       : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

The name associated with both the ENTITY and END keywords must match!

VHDL Code Explanation: Entity

- A simple example




```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                    f       : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

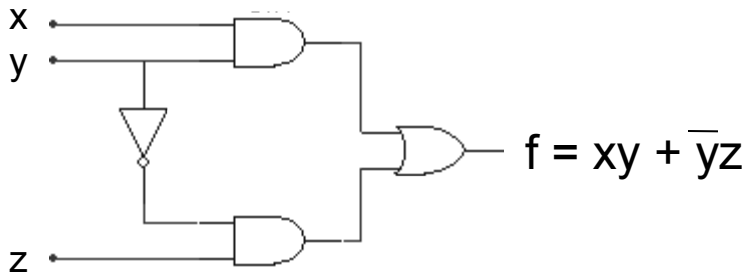
The ENTITY construct is analogous to a symbol in a schematic diagram

Names for ports, entities, architectures, etc:

1. May contain letters, numbers or the underscore (_)
2. Must start with a letter
3. Cannot be a keyword

VHDL Code Explanation: Entity

- A simple example



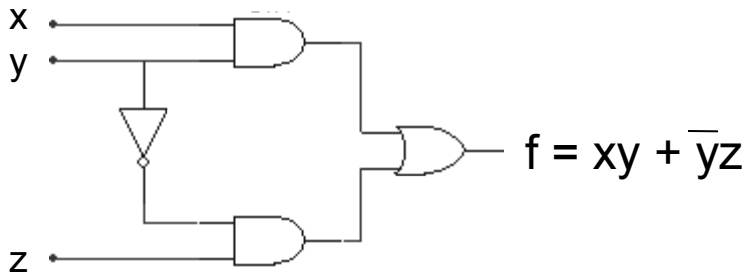

```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                      f     : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ENTITY construct is analogous to a symbol in a schematic diagram

All command or declaration lines must end with a semicolon (;)

VHDL Code Explanation: Architecture

- A simple example

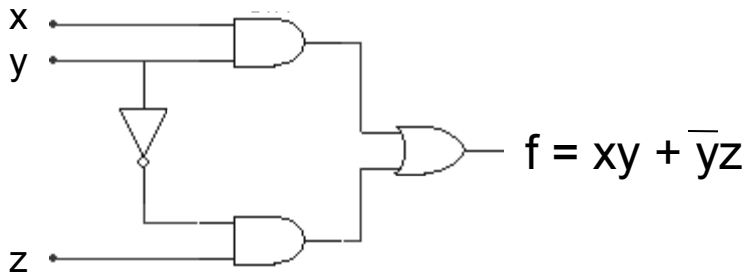



```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                      f     : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ARCHITCURE construct is analogous to the logic circuitry inside the symbol

VHDL Code Explanation: Architecture

- A simple example

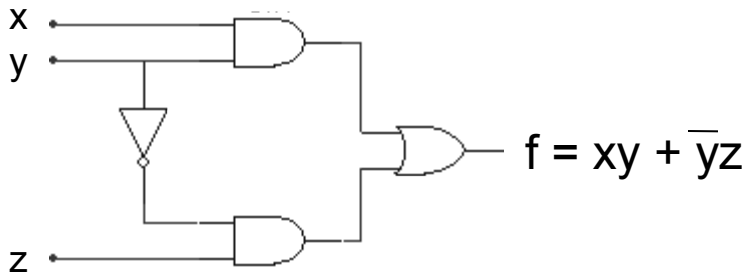



```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                        f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The ARCHITECTURE construct must be named: in this case “LogicFunc”

VHDL Code Explanation: Architecture

- A simple example

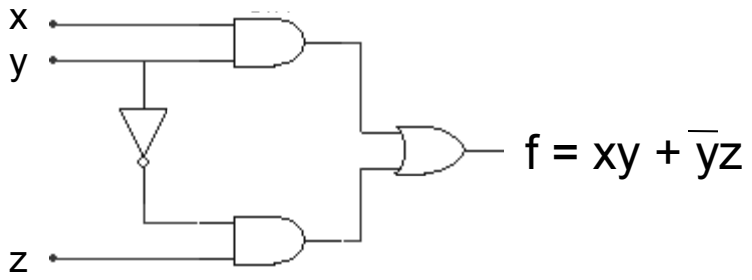



```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                        f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The statement could be read: “The architecture ***(name)*** of ***an entity*** is:”

VHDL Code Explanation: Architecture

- A simple example

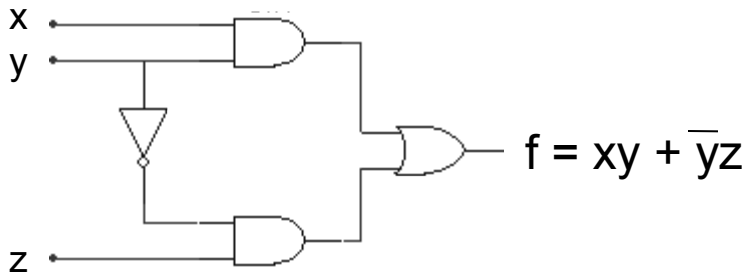



```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                             f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The architecture has a beginning and an end. The beginning is marked by the keyword: **BEGIN.** The end is marked by the keyword **END**

VHDL Code Explanation: Architecture

- A simple example

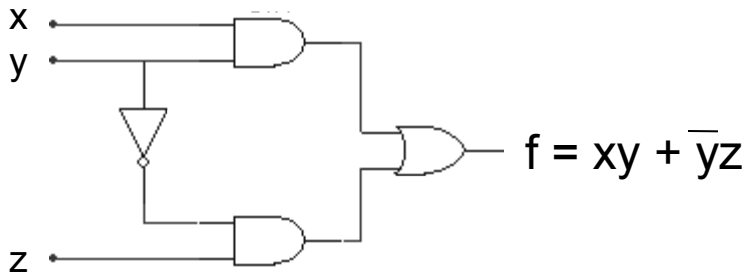



```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                            f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

The name associated with both the ARCITECTURE and END keywords must match!

VHDL Code Explanation: Architecture

- A simple example

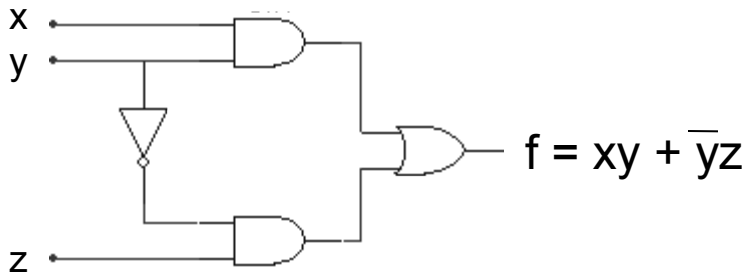



```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                            f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

f has been defined as an output

VHDL Code Explanation: Architecture

- A simple example

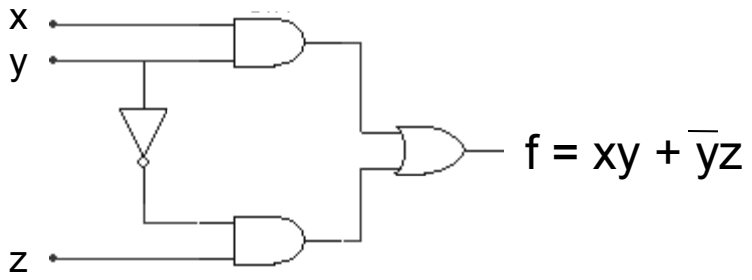



```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                        f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

and f should be assigned the result of the logic expression

VHDL Code Explanation: Architecture

- A simple example

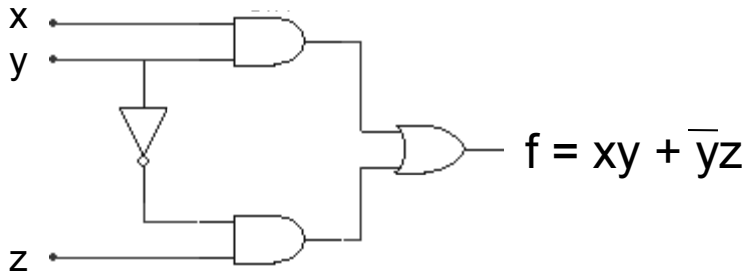



```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                             f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

"<=" is the assignment operator, specifying that f should be assigned this result

VHDL Code Explanation: Architecture

- A simple example

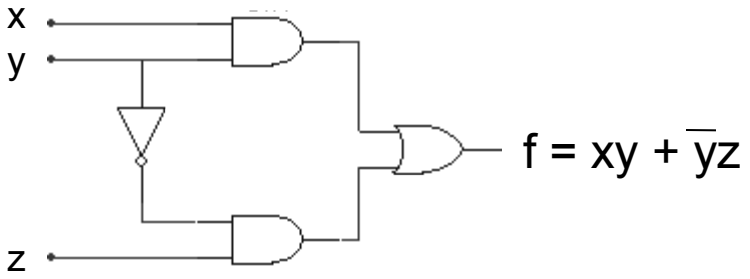



```
ENTITY  example1 IS
        PORT ( x, y, z       : IN   BIT ;
                             f      : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

AND, OR, NOT, NAND, NOR, XOR, and XNOR are Boolean operators specified by VHDL

VHDL Code Explanation

- A simple example

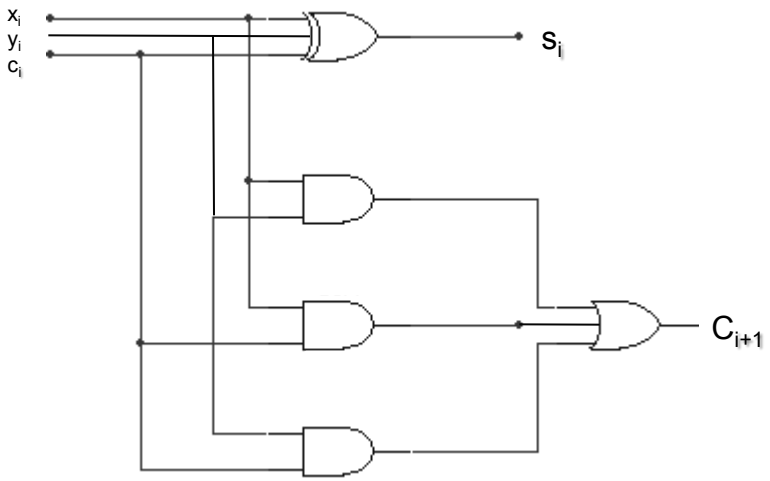



```
ENTITY  example1 IS
        PORT ( x, y, z      : IN   BIT ;
                      f     : OUT  BIT ) ;
END  example1 ;

ARCHITECTURE LogicFunc OF example1 IS
BEGIN
        f <= (x AND y) OR (NOT y AND z) ;
END LogicFunc ;
```

So, the ENTITY construct defines the inputs and outputs of a black box and the ARCHITECTURE construct the logic inside the black box

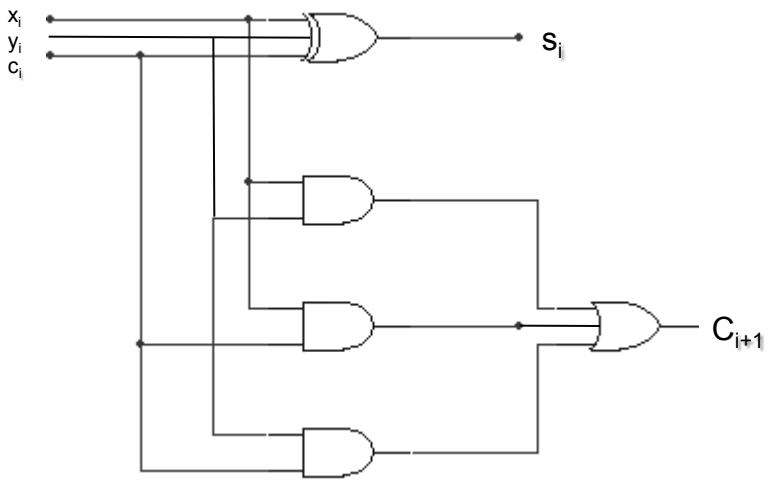
VHDL Example #2

- Exercise:
  - Take a few moments to write the VHDL code for the Full Adder shown below



VHDL Example #2

```
ENTITY  example1 IS
        PORT ( Xi, Yi, Ci       : IN   BIT ;
                        Si, Cout     : OUT   BIT ) ;
END  example1 ;

ARCHITECTURE FullAdder OF example1 IS
BEGIN
        Si <= Xi XOR Yi XOR Ci ;
        Cout<= (Xi AND Yi) OR (Xi AND Ci) OR (Yi AND Ci);
END FullAdder ;
```



Standard VHDL

- In previous examples, each signal was of type BIT
- This can be a problem because VHDL is a strong-typed language
  - Signals on both sides of the assignment operator (<=) must be the same type
- Most VHDL programs use a standard signal type called STD_LOGIC
  - Provides more flexibility
  - Meant to serve as the standard data type for logic signals




Standard VHDL

- To use the STD_LOGIC type, you must include the “LIBRARY” and “USE” directives at the beginning of the file (**1st two lines in the file**)
  - Standard for most VHDL
  - Necessary because:
    - Original VHDL (IEEE 1076 standard) did not include STD_LOGIC
    - New IEEE standard (IEEE 1164) does include STD_LOGIC
      - But VHDL does not, so we have to declare that the code will make use of this library (LIBRARY Directive)
      - The library defines the STD_LOGIC data type and specifies legal uses
        - » Such as Boolean Operations
        - » USE directive tells VHDL to make use of this file

```
LIBRARY ieee;
USE ieee.std_logic_1164.all
```

Standard VHDL Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all

ENTITY func1 IS
        PORT ( x1, x2, x3 : IN      STD_LOGIC ;
                            f  : OUT  STD_ LOGIC ) ;
END func1 ;

ARCHITECTURE LogicFunc OF func1 IS
BEGIN
        f <= (NOT x1 AND NOT x2 AND NOT x3) OR
                (NOT x1 AND x2 AND NOT x3) OR
                (x1 AND NOT x2 AND NOT x3) OR
                (x1 AND NOT x2 AND x3) OR
                (x1 AND x2 AND NOT x3) ;
END LogicFunc ;
```

The only differences between this and previous files are:
1. The inclusion of the 1st two lines and
2. The substitution of STD_LOGIC for BIT




Standard VHDL Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all

ENTITY func1 IS
        PORT ( x1, x2, x3 : IN      STD_LOGIC ;
                            f  : OUT  STD_ LOGIC ) ;
END func1 ;

ARCHITECTURE LogicFunc OF func1 IS
BEGIN
        f <= (NOT x1 AND NOT x2 AND NOT x3) OR
                (NOT x1 AND x2 AND NOT x3) OR
                (x1 AND NOT x2 AND NOT x3) OR
                (x1 AND NOT x2 AND x3) OR
                (x1 AND x2 AND NOT x3) ;
END LogicFunc ;
```

The only differences between this and previous files are:

1. The inclusion of the 1st two lines and
2. The substitution of STD_LOGIC for BIT

After compiling this file (in this case Quartus II), the “Analysis and Synthesis” report showed an optimized equation:

f = (X1 AND NOT X2) OR NOT X3

Standard VHDL Example

```
LIBRARY ieee;
USE ieee.std_logic_1164.all

ENTITY func1 IS
        PORT ( x1, x2, x3 : IN     STD_LOGIC ;
                            f  : OUT  STD_ LOGIC ) ;
END func1 ;

ARCHITECTURE LogicFunc OF func1 IS
BEGIN
        f <= (NOT x1 AND NOT x2 AND NOT x3) OR
                (NOT x1 AND x2 AND NOT x3) OR
                (x1 AND NOT x2 AND NOT x3) OR
                (x1 AND NOT x2 AND x3) OR
                (x1 AND x2 AND NOT x3) ;
END LogicFunc ;
```

The only differences between this and previous files are:
1. The inclusion of the 1st two lines and
2. The substitution of STD_LOGIC for BIT

After compiling this file (in this case Quartus II), the "Analysis and Synthesis" report showed an optimized equation:
f = (X1 AND NOT X2) OR NOT X3

The truth table (Look Up Table: LUT) generated for an FPGA is shown below:

X1	X2	X3	f
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0




Design Exercises

1. Design a logic circuit whose output is HIGH *only* when a majority of inputs A, B, and C are LOW.
2. A four-bit binary number is represented as $A_3\,A_2\,A_1\,A_0$, where A_3, A_2, A_1, and A_0 represent the individual bits and A_0 is equal to the LSB. Design a logic circuit that will produce a HIGH output whenever the binary number is greater than 0010 and less than 1000.

Design Exercises

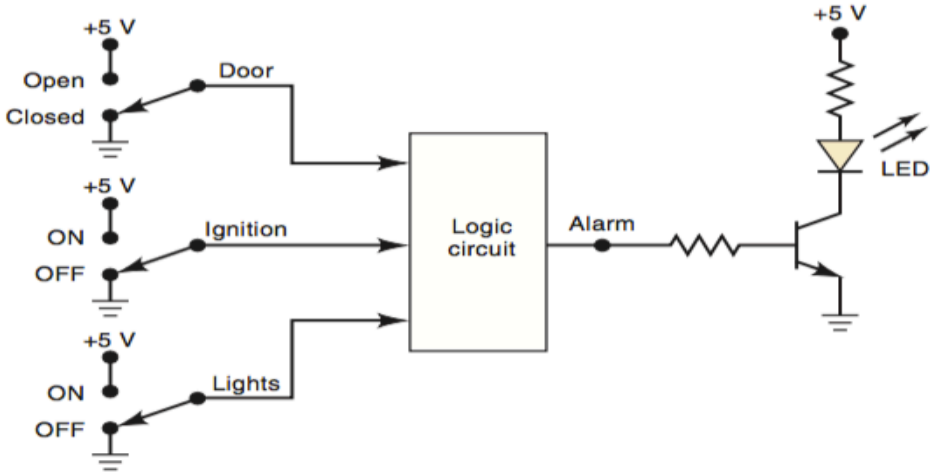
3. A manufacturing plant needs to have a horn sound to signal quitting time. The horn should be activated when either of the following conditions is met:

   i. It's after 5 o'clock and all machines are shut down.

   ii. It's Friday, the production run for the day is complete, and all machines are shut down.

   Design a logic circuit that will control the horn. (*Hint:* Use four logic input variables to represent the various conditions; for example, in- put *A* will be HIGH only when the time of day is 5 o'clock or later.)

FIGURE 4-66 Problem 4-8.



4. Figure 4-66 shows a diagram for an automobile alarm circuit used to detect certain undesirable conditions. The three switches are used to indicate the status of the door by the driver's seat, the ignition, and the headlights, respectively.

   Design the logic circuit with these three switches as inputs so that the alarm will be activated whenever either of the following conditions exists:

   i. The headlights are on while the ignition is off.

   ii. The door is open while the ignition is on.

Design Exercises

5. Four large tanks at a chemical plant contain different liquids being heated. Design a logic circuit that will detect whenever the level in tank *A* or tank *B* is too high at the same time that the temperature in either tank *C* or tank *D* is too low, given the following conditions:
   i. Liquid-level sensors are being used to detect whenever the level in tank *A* or tank *B* rises above a predetermined level.
   ii. Temperature sensors in tanks *C* and *D* detect when the temperature in either of these tanks drops below a prescribed temperature limit.
   iii. Assume that the liquid-level sensor outputs *A* and *B* are LOW when the level is satisfactory and HIGH when the level is too high.
   iv. Also, the temperature-sensor outputs *C* and *D* are LOW when the temperature is satisfactory and HIGH when the temperature is too low.