EEE205 - Digital Electronics (II)

Lecture 3

Xiaoyang Chen, Jiangmin Gu, and Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

In This Session

- More Powerful PLDs
  - Complex Programmable Logic Devices (CPLD)
  - Field Programmable Gate Arrays (FPGA)

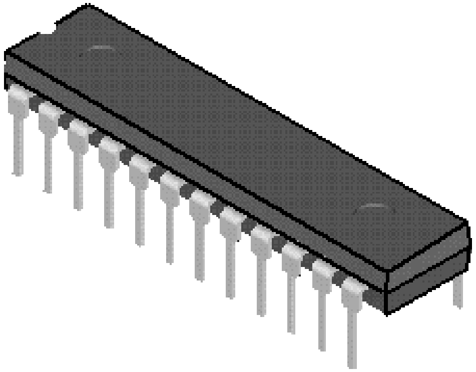
Programmable Logic Devices (PLDs)

Types

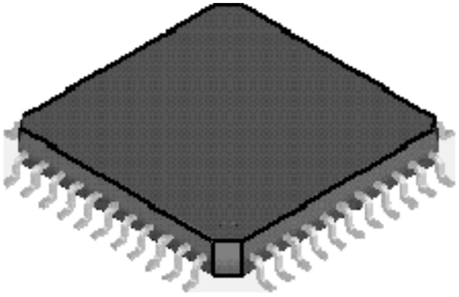
- SPLDs — up to 600 *equivalent gates* each.
- CPLDs — up to thousands of *equivalent gates* each.
- FPGAs — hundreds of thousands of *equivalent gates* each.

  equivalent gates: 2-input NAND gates

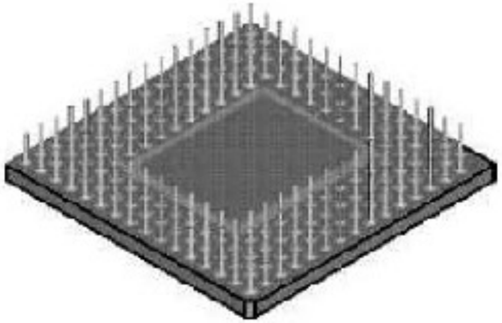
SPLD



CPLD



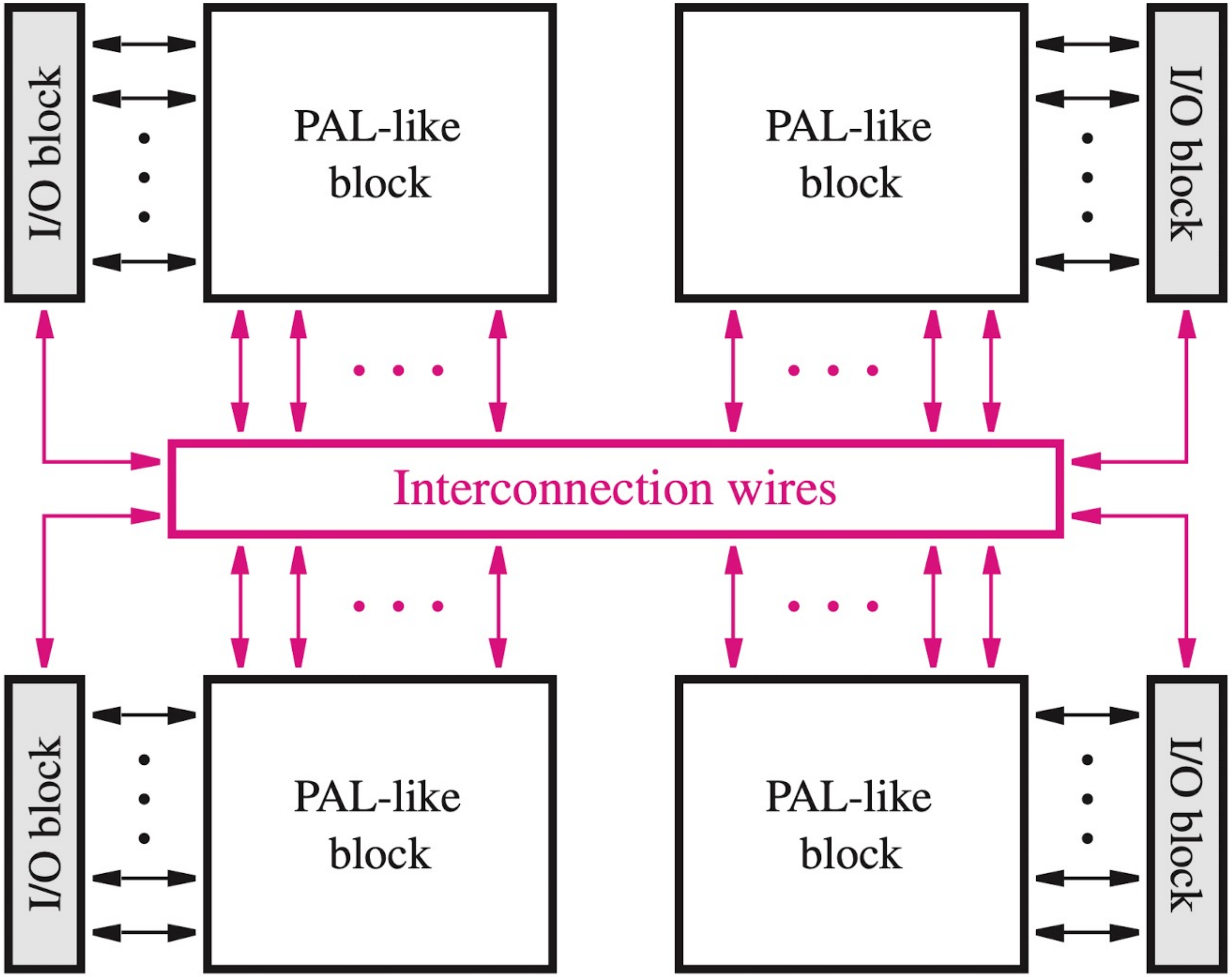
FPGA



Complex Programmable Logic Devices (CPLD)

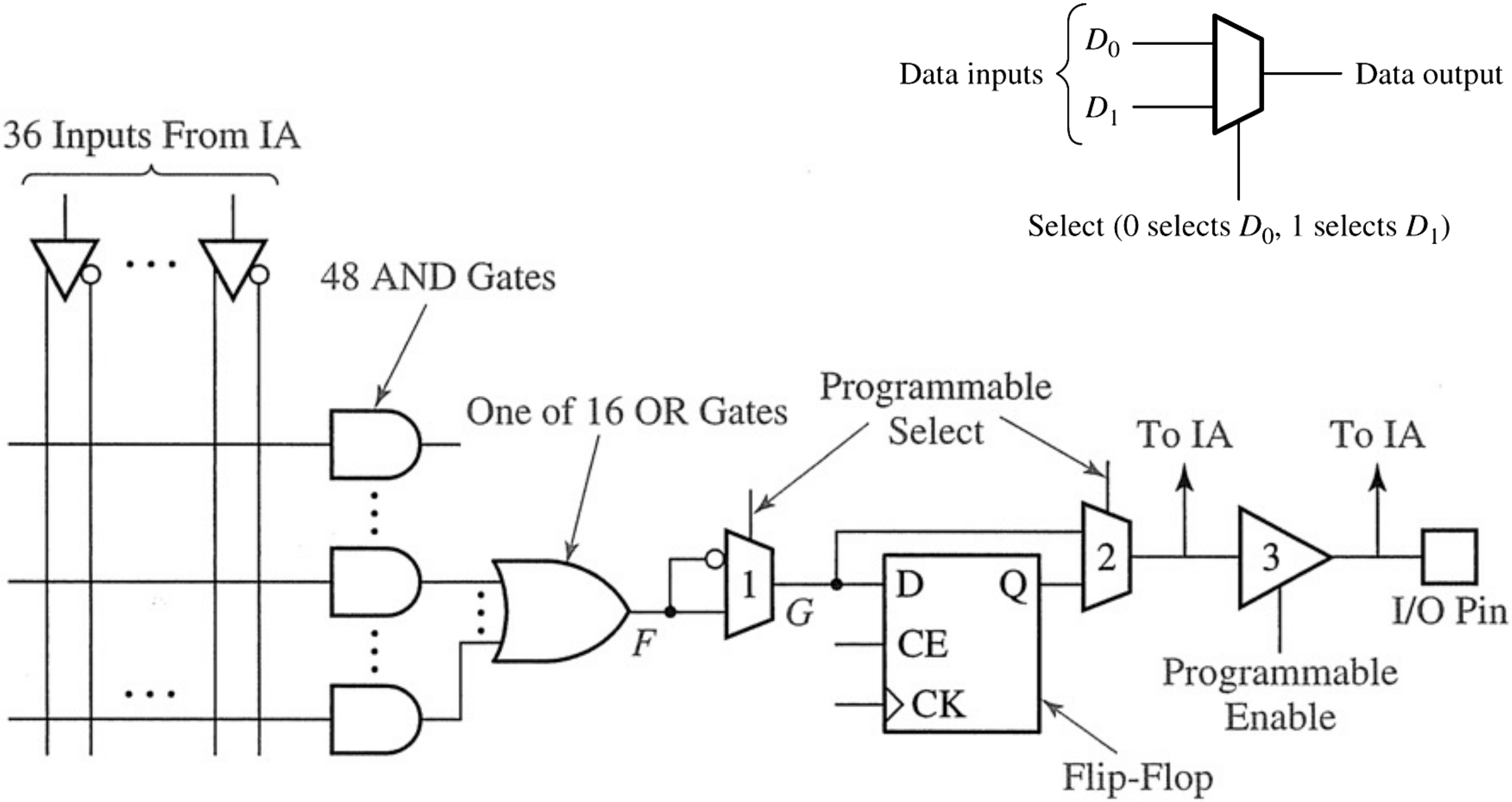
- A CPLD comprises multiple circuit blocks, each is similar to a PLA or a PAL and called a *PAL-like block*.
- These PAL-like blocks are connected to a set of *interconnection wires*.
- Each PAL-like block is also connected to a subcircuit called *I/O block*, which is attached to a number of the chip’s input and output pins.

Complex Programmable Logic Devices (CPLD)



Complex Programmable Logic Devices (CPLD)

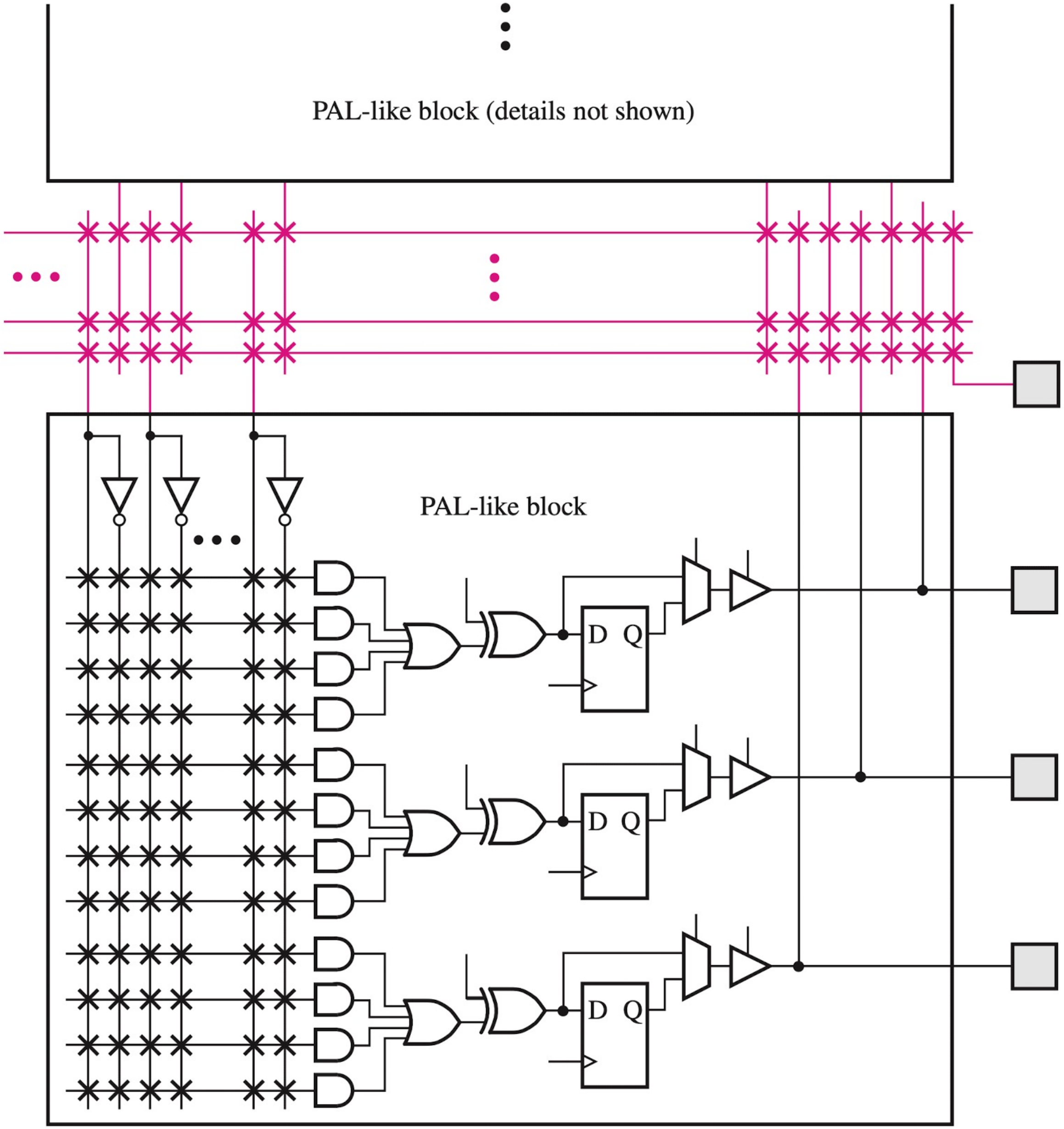
- Each PAL-like block includes multiple (typically 16) macrocells.



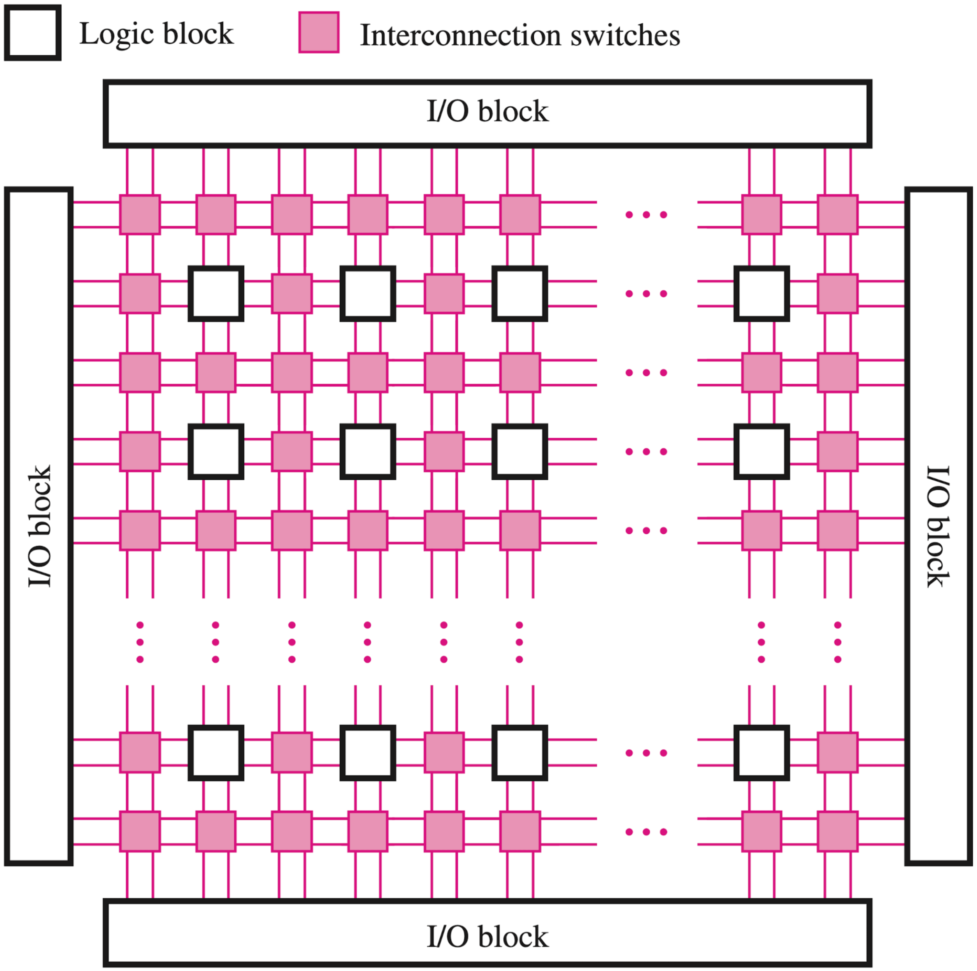
CPLDs

Example:

An I/O pin may be used as an input - the buffer must be disabled

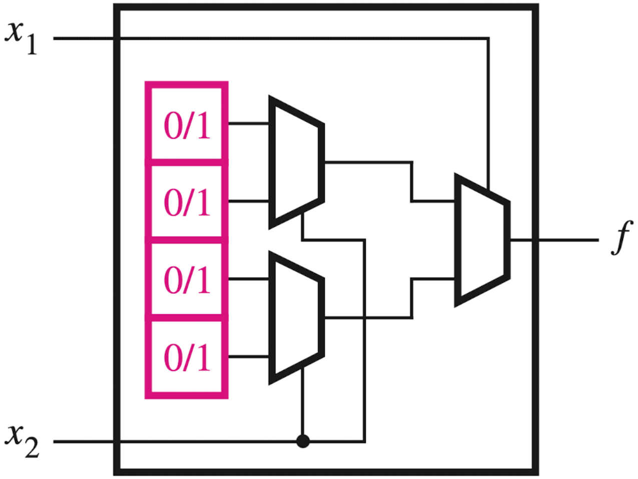


Field Programmable Gate Arrays (FPGA)



- An FPGA consists of an array of configurable logic blocks (CLBs).
- Interconnects are available between the CLBs.
- The CLBs are surrounded by I/O blocks.

Field Programmable Gate Arrays (FPGA)



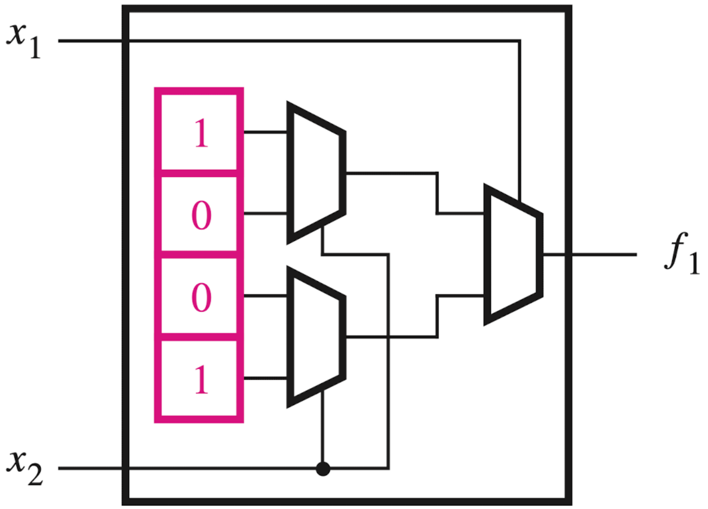
Configuration Logic Block (CLB):

- It often consists of an **LUT (lookup table)** that can generate any logic function — a truth-table based approach.
- Different from **AND-OR arrays** in SPLDs and CPLDs — an SOP Boolean-expression based approach.

Field Programmable Gate Arrays (FPGA)

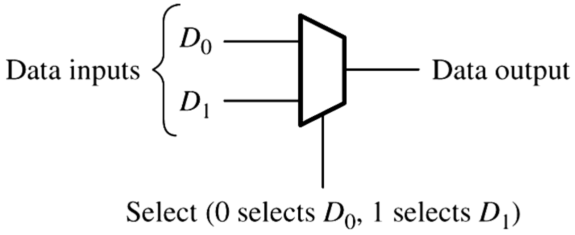
x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

$$f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$$



- An LUT contains storage cells, each stores an entry of the truth table.
- The outputs of such cells are multiplexed with the input variables as the data select.

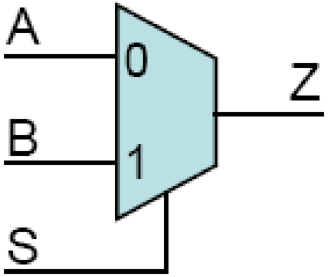
LUTs usually have 4-5 inputs in commercial FPGAs.



Field Programmable Gate Arrays (FPGA)

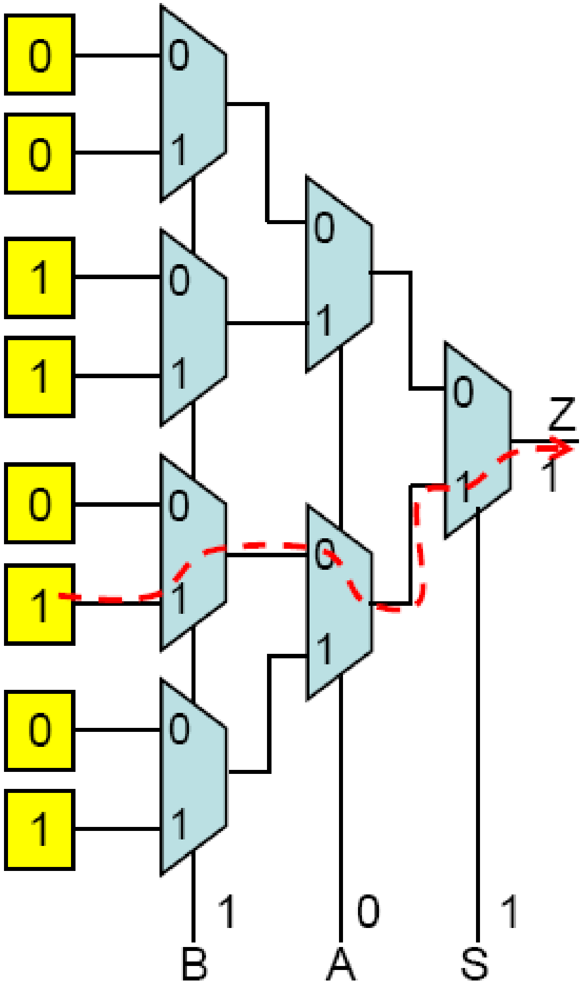
Example: an LUT to implement a multiplexer

Multiplexer

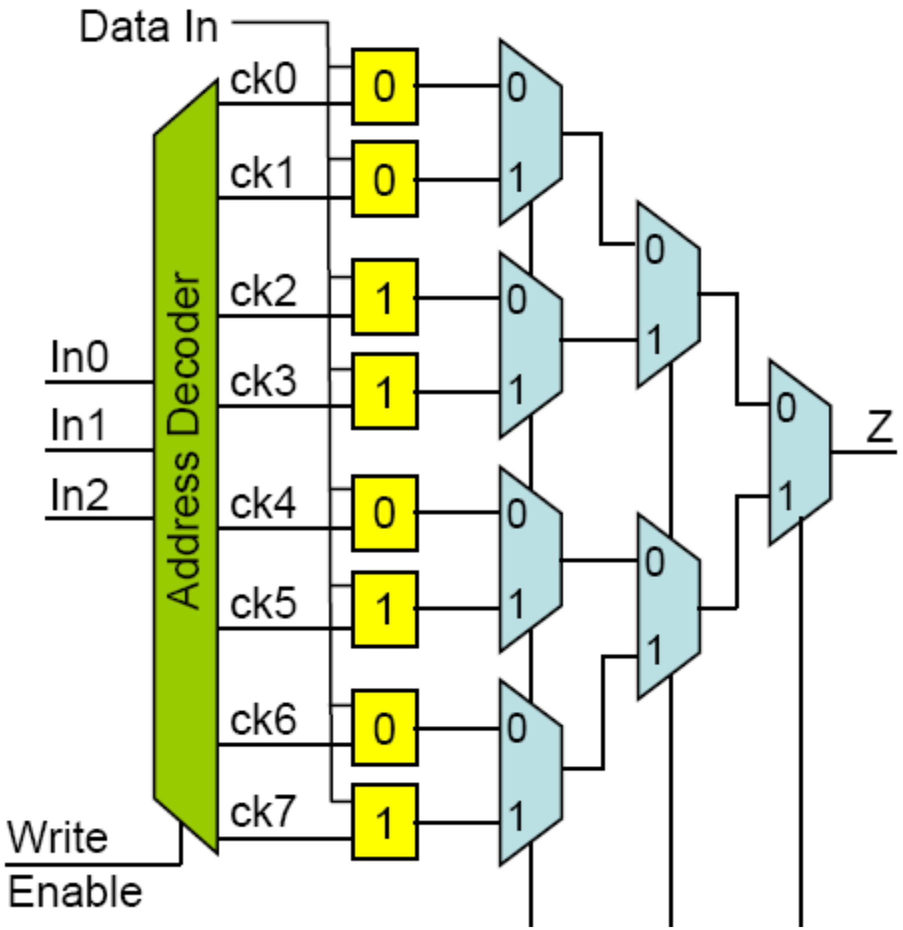


Truth table

S	A	B	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



Field Programmable Gate Arrays (FPGA)



1. Normal LUT mode performs read operations.
2. Address decoder with write enable generates clock signals to latches for write operations.

Field Programmable Gate Arrays (FPGA)

A Programmed FPGA

- Each switch (X) in blue is turned on and each switch in black is turned off.
- The top row implements the function $f_1 = x_1x_2$ and $f_2 = \overline{x_2}x_3$.
- The bottom row produces $f = f_1 + f_2 = x_1x_2 + \overline{x_2}x_3$.

x_1	x_2	f_1
0	0	0
0	1	0
1	0	0
1	1	1

$f_1 = x_1 x_2$

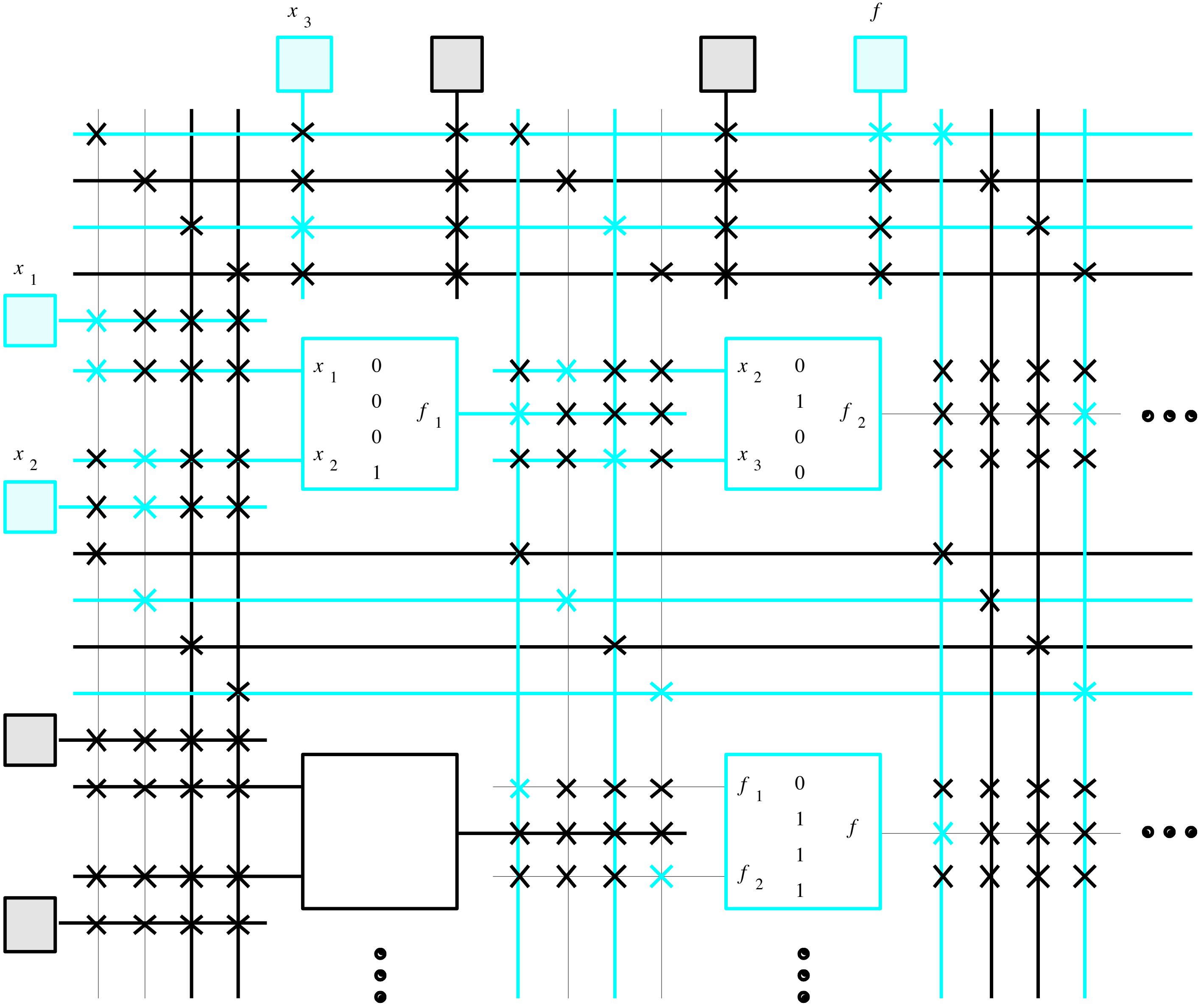
x_2	x_3	f_2
0	0	0
0	1	1
1	0	0
1	1	0

$f_2 = \overline{x_2} x_3$

f_1	f_2	f
0	0	0
0	1	1
1	0	1
1	1	1

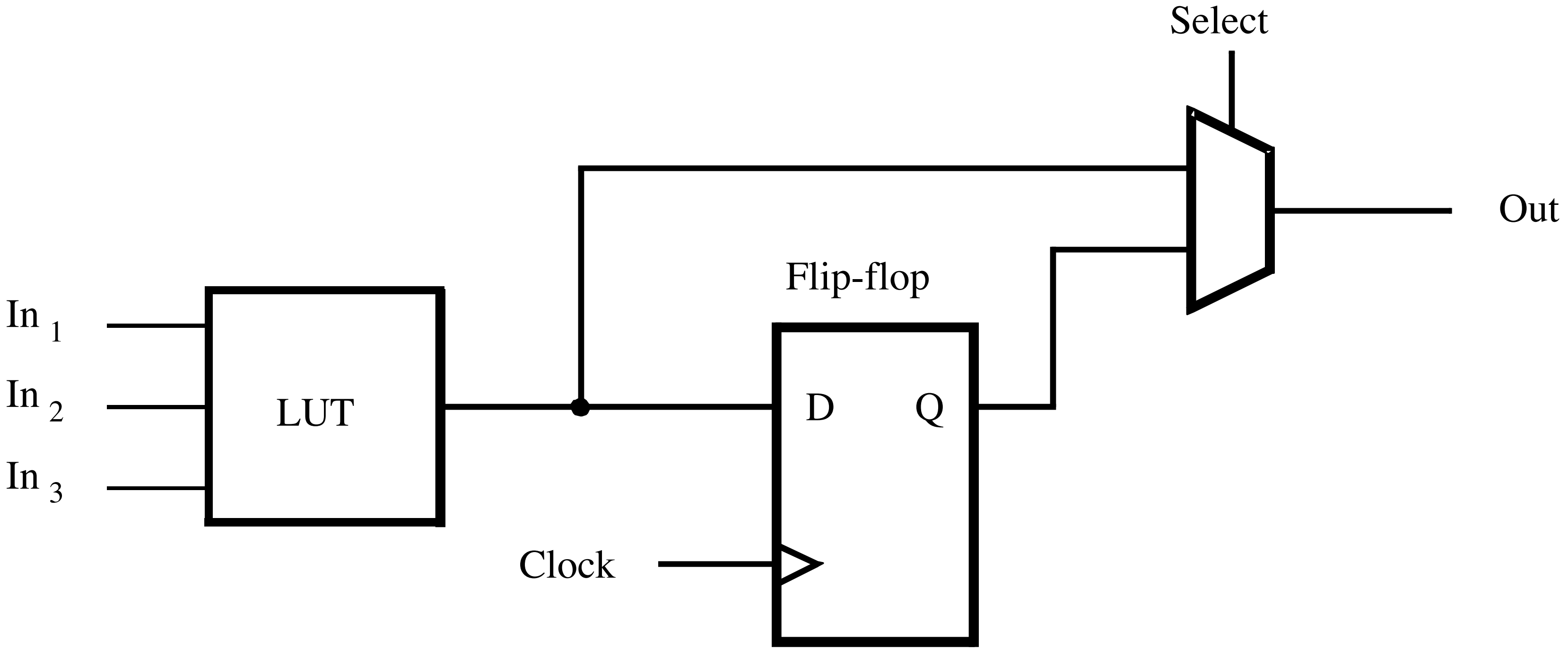
$f = f_1 + f_2$

Field Programmable Gate Arrays (FPGA)



Field Programmable Gate Arrays (FPGA)

Flip-flops may be included in an FPGA logic block to facilitate registered outputs.



EEE205 – Digital Electronics (II)

Lecture 4

Xiaoyang Chen, Jiangmin Gu, and Ming Xu

Dept of Electrical & Electronic Engineering

XJTLU

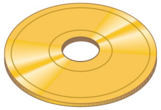
In This Session

- PLD Programming
- Hardware Description Languages (HDLs)

PLD Programming

The three components required:

- A computer
- A CAD system
- A programmer or a JTAG interface that is connected with the computer through a cable.



FPGA CPLD and ASIC from Altera - Windows Internet Explorer

http://www.altera.com/

Language: English [Change]

Manage Multiple Protocols with One Networking Kit

Learn to manage multiple protocols with One Networking Kit

Pre-order BeMicro SDK Motor Control Kit

Buy $75 MAX® V CPLD Development Kit

What's New

Altera Showcases FPGAs for Enterprise Flash Memory Solutions

Altera's New Dynamic Packaging Datasheet Simplifies Packaging Selection for Altera Devices

Quartus II Software Version 11.0 Service Pack 1 is Available for Download Today

Altera offers three ways to offset your costs by providing discounts on its technical training

Altera's External Memory Interface

Download Software

- Quartus II Web Edition (Free)
- Quartus II Subscription Edition
- ModelSim®-Altera
- Nios® II Embedded Design Suite
- DSP Builder

Quartus II downloads include MegaCore IP Library

Get a License

- Self-Service Licensing Center

License Support

- Licensing FAQ
- Licensing Webcast
- Video Tutorials

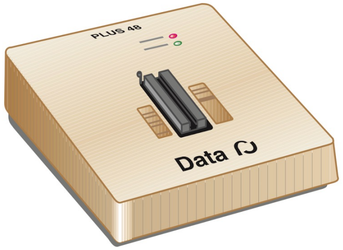
Software Support

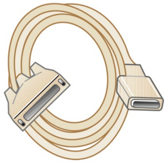
- Quartus II Support

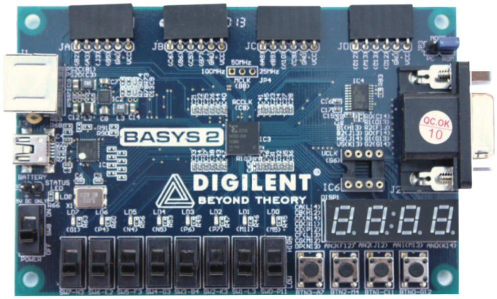
Learn Four Reasons FPGAs are Right for Motor Control

Watch the Video
Products | End Markets | Technology | Training | Support | About Altera | Buy Online
Jobs | Investor Relations | Contact Us | Site Map | Privacy | Legal Notice
Altera Forum
Altera Wiki
Email Updates
Follow Us
Internet | Protected Mode: On
80%

Copyright © 1995-2011 Altera Corporation. All Rights Reserved.







PLD Programming

The CAD System

- Design entry
- Synthesis and optimization
- Simulation
- Physical design
- Programming

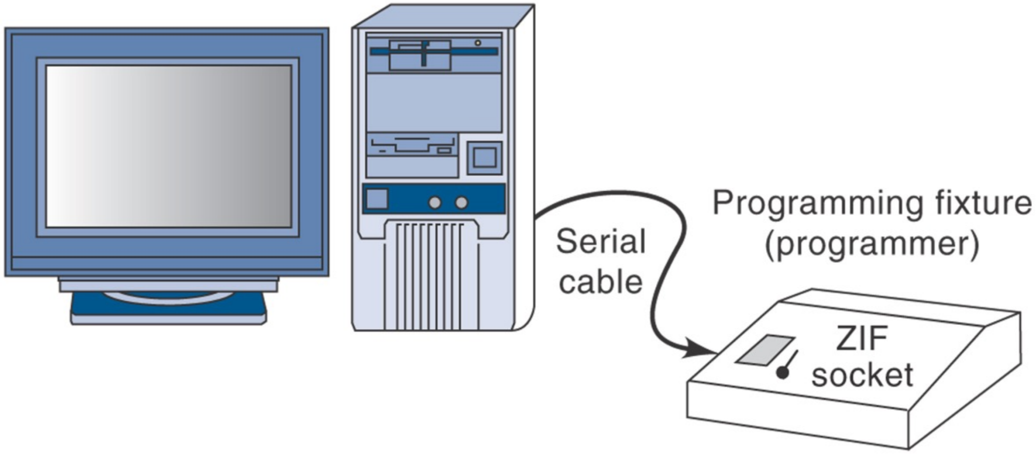
Examples

- Xilinx ISE
- Altera Quartus II
- Altera MAX+plus II

PLD Programming

Programming by a Programmer

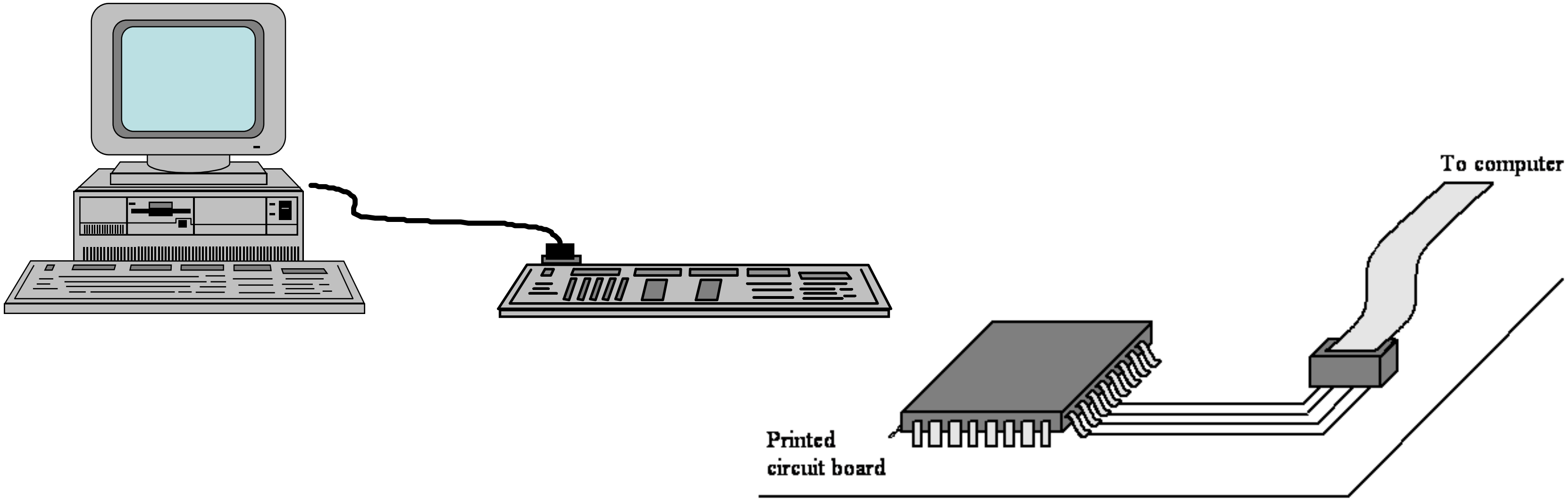
- The PLD is removed from its circuit board and placed into a programmer for configuration
- It is mainly used for SPLDs.



PLD Programming

In-System Programming (ISP)

- The PLD is attached to its circuit board while being programmed through a 4-wire JTAG port.
- There must be dedicated pins on the PLD for the JTAG port.
- It is mainly used for CPLDs and FPGAs.



PLD Programming

The design entry methods may include:

- Schematic Capture
- Truth Tables
- Hardware Description Languages (HDLs)

An Overview of HDLs

- A **H**ardware **D**escription **L**anguage (HDL) is a computer language that is used to describe hardware.
- Unlike schematic capture, HDL based designs are highly portable and independent of technology.

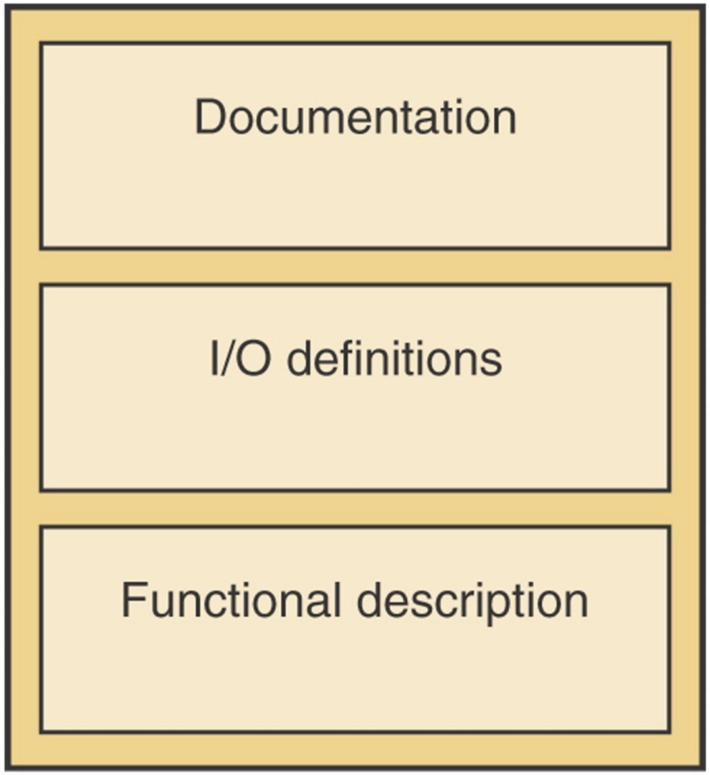
An Overview of HDLs

Examples

- Verilog HDL (**Veri**fy **Log**ic, IEEE standard)
- VHDL (**V**ery High Speed Integrated Circuit HDL, IEEE standard)
- AHDL (**A**ltera HDL)
- ABEL
- CUPL

AHDL will be introduced in this module.

AHDL Overview

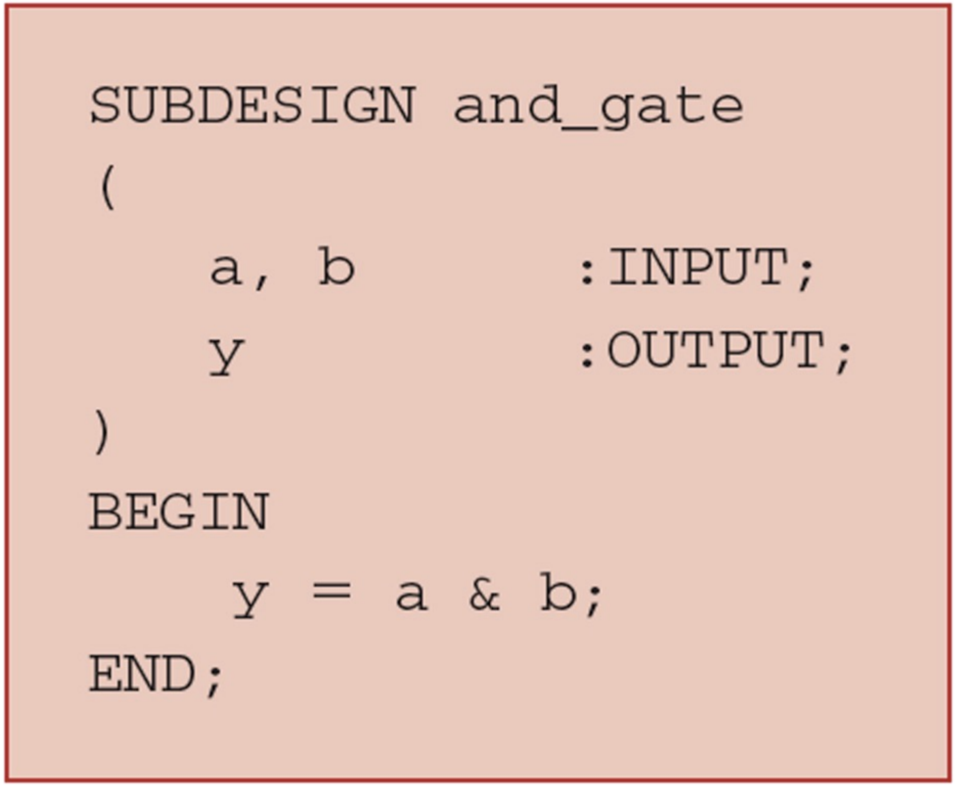


Format of AHDL files:

- Documentation description (naming).
- I/O definitions
  - Mode – whether a port is input, output, or both.
  - Type – the number of bits
- Functional description is the definition of the circuit's operation.

AHDL Overview

Example: an AND gate



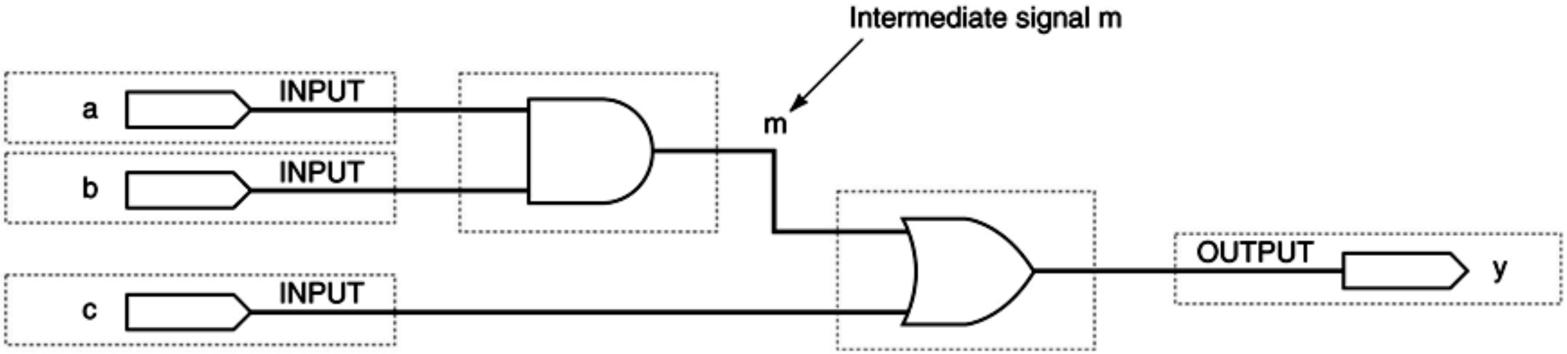
```
SUBDESIGN and_gate
(
    a, b          :INPUT;
    y             :OUTPUT;
)
BEGIN
     y = a & b;
END;
```

AHDL Overview

- **SUBDESIGN** names the circuit block, in this case: and_gate
- The definitions of inputs (**INPUT**), outputs (**OUTPUT**) or bidirectional port (**BIDIR**) are enclosed in parenthesis.
- The description of operation is between the **BEGIN** and **END** keywords.
- Boolean operators: **!** For NOT, **&** for AND, **#** for OR, and **$** for XOR.

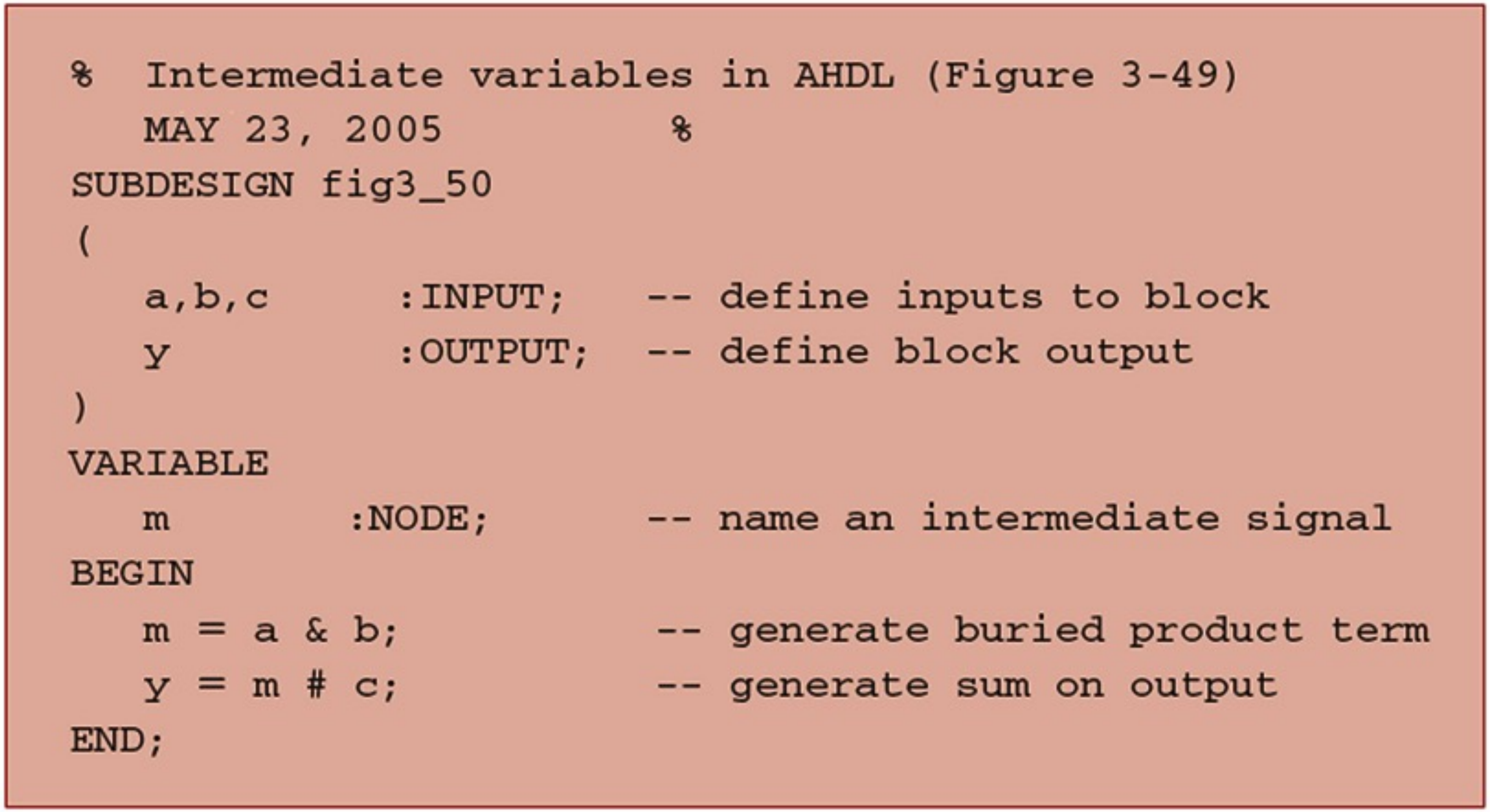
Intermediate Signals

- There are points in the circuit that are neither inputs nor outputs for the block.
- They are available only within this block rather than to other block.
- They are called **buried nodes** by AHDL.



Intermediate Signals

AHDL Buried Nodes



```
%  Intermediate variables in AHDL (Figure 3-49)
   MAY 23, 2005         %
SUBDESIGN fig3_50
(
   a,b,c     :INPUT;    -- define inputs to block
   y         :OUTPUT;   -- define block output
)
VARIABLE
   m       :NODE;       -- name an intermediate signal
BEGIN
   m = a & b;            -- generate buried product term
   y = m # c;            -- generate sum on output
END;
```

Intermediate Signals

AHDL Buried Nodes

- Comments are enclosed by a pair of **%** characters or after two dashes.
- The optional **VARIABLE** Section is used to declare and/or generate variables, e.g. buried nodes, primitives, etc.
- NODE designates the mode of the variable. Other modes include TRI_STATE_NODE, (primitives) LATCH, DFF, JKFF.

AHDL Overview

AHDL vs. Programming Languages

- All the statements between BEGIN and END are evaluated constantly and **concurrently** (at the same time).
- The order in which they are listed makes no difference.
- In the contrast, a computer programming language follows instructions in **sequential** order.