Programmable Logic Device (PLD)

Today, complicated digital circuits are implemented using programmable logic devices.

These devices are integrated circuits that include many reconfigurable logic gates. (From several hundred to several million).

Some PLDs also include memory units (flip-flops).

The designer can reconfigure the connections between logic gates in the PLD using a programming language and a programming device.

It is possible to implement complicated digital circuits with only a single IC (PLD).

There are different kinds of PLDs:

- *Programmable Logic Array* – **PLA**
- *Programmable Array Logic* – **PAL®**
- *Generic Array Logic* – **GAL**
- *Complex PLD* – **CPLD**
- *Field-Programmable Gate Array* – **FPGA**

PAL is a registered trademark of Lattice Semiconductor Corp.

Programming of PLDs:

In early versions of PLDs (PLA, PAL), bipolar transistors were used (See Chapter 11).
They have fuses on the connection points between gates, which provide reconfiguration (programming) of devices.
In these devices, fuses can be blown only once; therefore, they are called "one-time programmable (OTP)."

Today's devices (GAL, CPLD, FPGA) are made of CMOS transistors and contain memory units for programming.
They can be erased and reprogrammed many times.

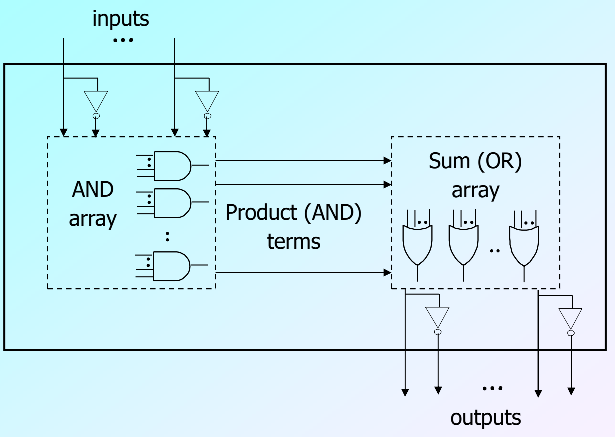
To program PLDs, various Hardware Description Languages (**HDL**) and programming devices are used.

Some examples of HDLs:

- PALASM
- ABEL
- Verilog
- VHDL (*Very high speed integrated circuits* HDL)

Programmable Logic Array - PLA

A PLA has AND (product) units in the input layer and OR (sum) units in the output layer.



AND and OR arrays can both be flexibly programmed.

Programmable Logic Array - PLA (contd)

Parameters that determine the limitations of a PLA:

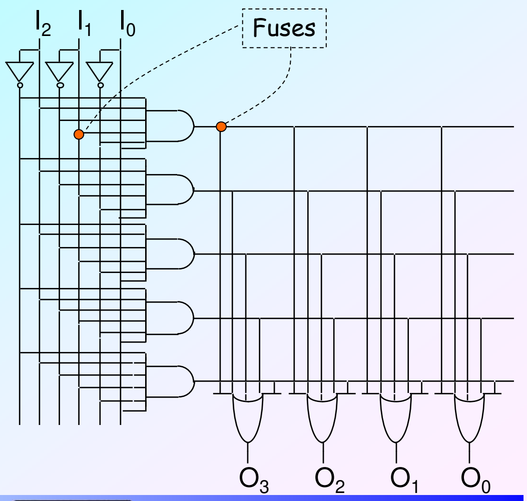
- Inputs : n
- Outputs : m
- AND gates (products) : p

Such a device is called "n x m PLA with p products".

On the right, the internal structure of a 3x4 PLA with 5 products is shown.

In fact, real PLAs include about 100 gates.

Example: 82S100
16 inputs, 8 outputs, 48 products

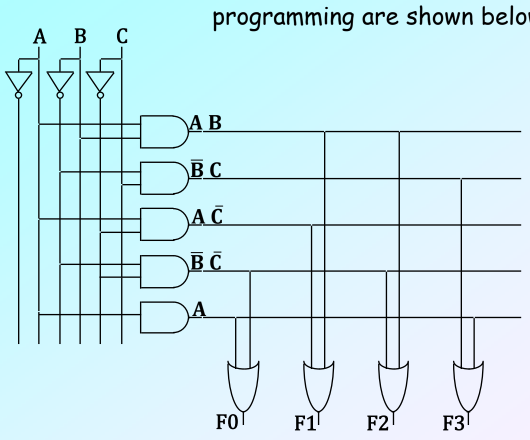


Example:

$F0 = A + \overline{B}\,\overline{C}$
$F1 = A\,\overline{C} + A\,B$
$F2 = \overline{B}\,\overline{C} + A\,B$
$F3 = \overline{B}\,C + A$

In the programming process, unnecessary fuses are blown.

Internal connections of a 3x4 PLA with 5 products after programming are shown below.



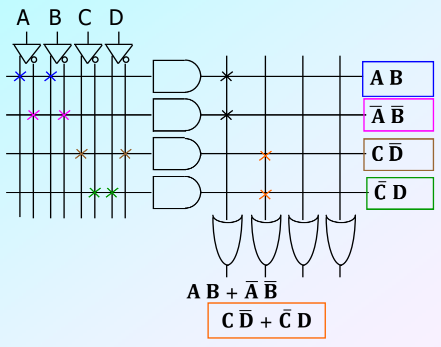
Simple Representation:

For the sake of simplicity, we do not show all of the connections on a PLA.

Instead, we put an 'X' on the connection points which are connected to the inputs of gates.

Example:

$F0 = A\,B + \overline{A}\,\overline{B}$
$F1 = C\,\overline{D} + \overline{C}\,D$



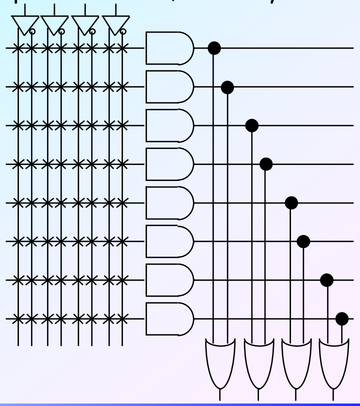
Programmable Array Logic - PAL

Inputs of AND gates can be flexibly programmed as in PLAs.

However, inputs of OR gates are fixed. To each OR gate, only outputs of certain AND gates can be connected.

For example, to the inputs of the first OR gate, only outputs of the first two AND gates can be connected.

PALs can be easily programmed, they are cheaper than PLAs, and they can contain more gates.



PALs were introduced by the company Monolithic Memories, Inc. (MMI).

MMI obtained a registered trademark on the term PAL for use in "Programmable Semiconductor Logic Circuits."

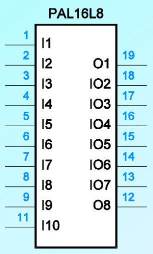
The trademark is currently held by Lattice Semiconductor Corporation.

MMI was acquired by Advanced Micro Devices (AMD).

Lattice Semiconductor then acquired the programmable logic division of AMD (Vantis).

Example: PAL16L8

- Pins on the left side and bottom of the logic diagram (pins 1 to 9 and pin 11) are used for inputs



- Pins 12 and 19 can be used only as outputs, but six of the outputs (pins 13 to 18) are also available as inputs via the feedback line connection after the inverting output buffer.
- This feature, called programmable I/O, lets the user program each of these six pins as either input or output.

PAL16L8:

- 16 signifies the maximum number of potential inputs (10 dedicated inputs and 6 programmable I/O).
- 8 signifies the number of outputs.
- L signifies the output type, which is active low for this PAL part type.

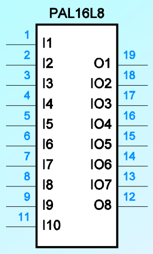
Example: A part of PAL 16L8 device is shown below:

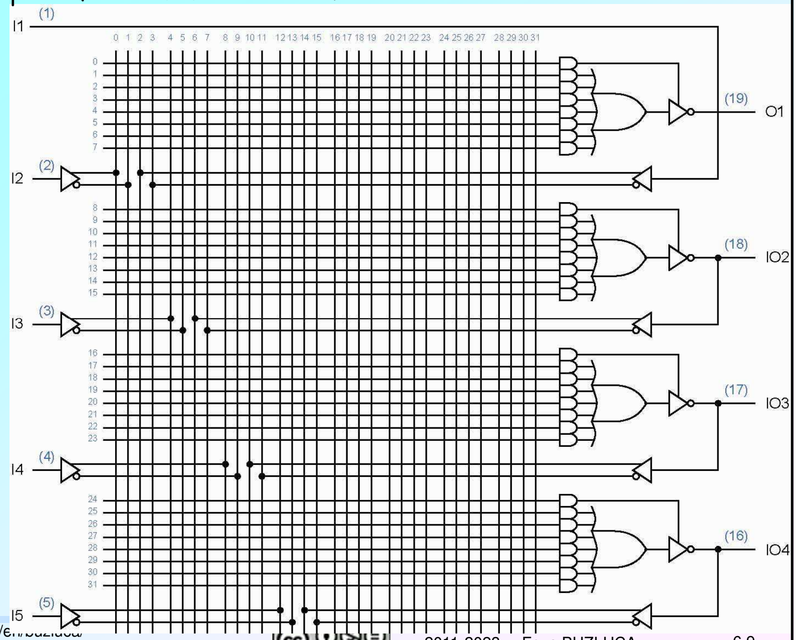
16 inputs,

8 outputs,

64 products (AND)

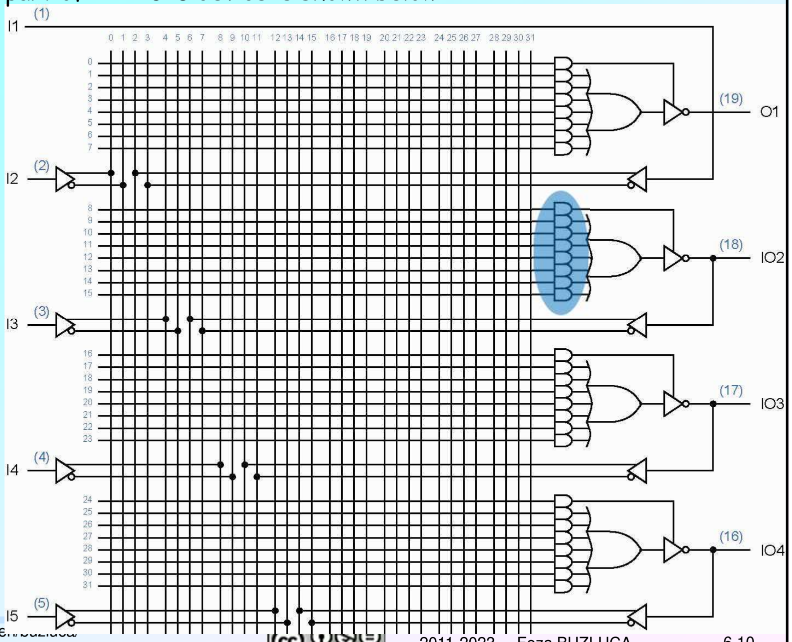
Each AND gate has 2x16 inputs (input variables and their complements).





Example: A part of PAL16L8 device is shown below:

- Eight AND gates are associated with each output pin.
  - Seven of them provide inputs to a fixed 7-input OR gate.
  - The eighth (called output-enable gate) is connected to the three-state enable input of the output buffer.
- The buffer is enabled only when the output-enable gate has an output of one.



Generic Array Logic - GAL

Its logical properties are similar to those of the PAL.

It is made up of CMOS transistors. It can be erased and programmed many times.

Lattice Semiconductor introduced it.

Example: GAL16V8

Complex PLD - CPLD

It contains several PLDs (*macrocells*).

Each internal PLD (macrocell) has GAL properties.

CPLDs typically have thousands to tens of thousands of logic gates.

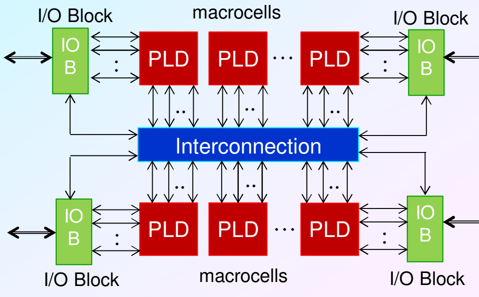
Internal structures of macrocells and connections between them can be programmed.

Example: Atmel (Microchip) ATF1500

32 input/output pins + 4 inputs pins

32 PLDs (*macrocells*).

Architecture of a CPLD:



Field-Programmable Gate Array - FPGA

It consists of an array of configurable logic blocks (CLBs) connected via programmable interconnects.

It can be erased and programmed many times.

They contain several thousand to several million logic gates.

They can be used to implement complex digital circuits, e.g., special-purpose microprocessors.

Compared to CPLDs, FPGAs are more flexible and can implement more complicated circuits.

However, the delay of FPGAs is higher.

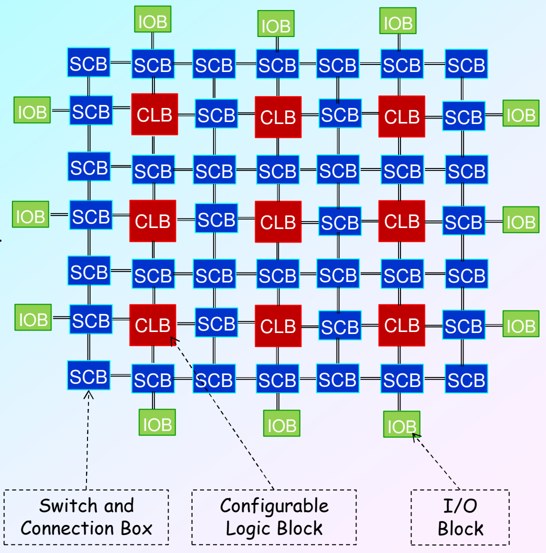
Example:

AMD Artix™ UltraScale+™ AU25P

304 input/output pins

308000 system logic cells

Architecture of an FPGA:



ASIC (Application-specific integrated circuit)

An application-specific integrated circuit (ASIC) is an integrated circuit (IC) designed for a particular use.

An ASIC is not a reprogrammable device like an FPGA.

During the design of an ASIC, functional blocks are taken from a library, interconnected, and verified via simulation.

ASICs often include entire microprocessors, memory blocks, and other blocks for I/O operations. Such an ASIC is called a **SoC** (**system-on-chip**).

HDLs are also used for designing ASICs, just like they are used for PLDs.

ASIC are used in medical image processing, encoding/decoding data in communication devices, controlling the charging process in smartphones, etc.

The non-recurring engineering (NRE) cost of an ASIC is very high.

- **Non-recurring engineering (NRE) cost** is the one-time cost to research, design, develop, and test a new product.
- **Production costs** must be paid constantly to maintain the production of a product.

Therefore, device manufacturers usually choose FPGAs for prototyping and low-volume production, while ASICs are preferred for high-volume production with amortized NRE costs.

HDL-Based Design

It is difficult to design (describe) large and complex systems using Boolean algebra and logic functions.

Similar to software development, most digital design is now done at higher levels of abstraction.

Software Design:

Although a CPU can only directly execute machine code, we do not use machine language or assembly language for large systems unless absolutely necessary.

We write complex programs using high-level programming languages such as C++, Java, C#, and Python, which are closer to human language than to machine language.

Compilers convert these programs to machine code.

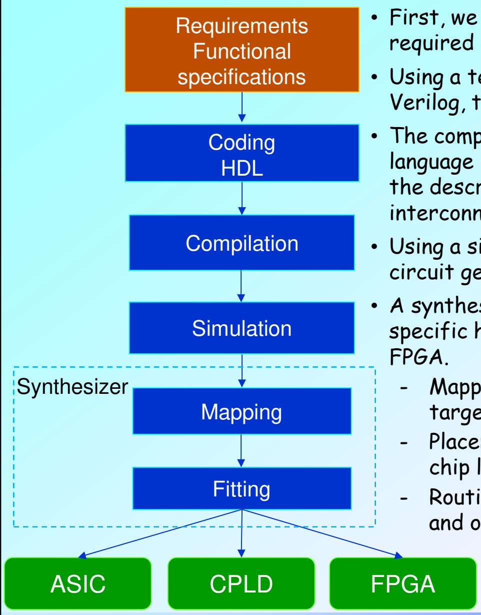
Hardware Design:

Designers use Hardware Description Languages (HDLs) such as Verilog or VHDL to describe circuits at an abstract level. They do not have to use Boolean algebra.

In addition, designers can obtain commonly used functions and subsystems under a license from an intellectual property (IP) provider to integrate with their custom circuit.

Finally, a Verilog or VHDL synthesis tool can produce a circuit by building an ASIC chip or programming an FPGA.

HDL-Based Design Flow



- First, we identify the functional specifications of the required circuit.
- Using a text editor, we write the program in an HDL, e.g., Verilog, that describes the specifications of the circuit.
- The compiler creates a file in a digital-design description language RTL (register-transfer language). The RTL file is the description of the logic operations and interconnections.
- Using a simulation tool, we can verify if the designed circuit generates the expected outputs for given inputs.
- A synthesizer tool is used to target the RTL design to a specific hardware technology such as an ASIC, CPLD, or FPGA.
  - Mapping the RTL design into hardware elements in the target technology.
  - Placement of the necessary elements onto a physical chip layout.
  - Routing: Finding or creating paths between the inputs and outputs of placed elements.

Verilog

Verilog is one of the popular hardware description languages.

The other one is VHDL.

The syntax of Verilog is similar to that of the C programming language.

Standards:

- Verilog-2001: IEEE standard 1364-2001.
- Verilog-2005: IEEE standard 1364-2005.
- System Verilog: IEEE standard 1800-2009.

The 2009 standard, System Verilog, includes Verilog-2001/2005 as a subset and introduces new features for specifying, designing, and verifying complex systems.

The basic unit of design and programming in Verilog is a **module**.
A module may correspond to a single piece of hardware.
Modules are similar to functions or procedures in programming.
Modules can be used as building blocks in other larger modules.

A Verilog module consists of *declarations* and *statements*.

Declarations describe the names and types of the module's inputs and outputs, as well as local signals, variables, and constants used internally in the module.

The statements specify or "model" the operation of the module's outputs and internal signals.

Verilog Examples

The Verilog examples in this course have been compiled and tested using the Xilinx (AMD) Vivado® suite.

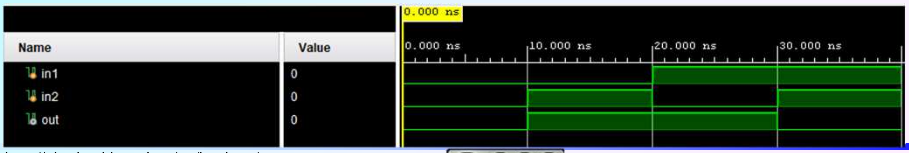
https://www.xilinx.com/products/design-tools/vivado.html

https://www.xilinx.com/support/download.html

Example: XOR Function

```
// Example: XOR Function
module XOR(
    input in1,                // Declarations
    input in2,
    output out
    );
    assign out = in1 && !in2 || !in1 && in2;   // Logic function of XOR
 endmodule
```

Simulation:



Example: Full Adder using XOR Module

Full Adder: Slide 5.3

```
// Example: Full Adder
  module FullAdder(
    input a,           // First number
    input b,           // Second number
    input Cin,         // Carry input
    output s,          // Output Sum
    output Cout        // Carry output
    );
    wire x;            // Internal wire

    // We use the XOR module designed in advance
    XOR XOR1(.in1(a), .in2(b), .out(x));               // x = a XOR b
    XOR XOR2(.in1(Cin), .in2(x), .out(s));             // s = Cin XOR x
    // Cout = a·Cin + b·Cin + a·b
    assign Cout = a && Cin || b && Cin || a && b;
 endmodule
```

Example:
`FullAdder.v`
`Adder_Test.v`

Example: 4-bit Prime number detector

In the previous examples, we wrote Boolean expressions directly into our source programs.
In this example, we write the program for a 4-bit prime number detector using `if-else` statements.
The abstraction level of this program is higher than the those of the previous ones.

```
module isPrime(
    input [3:0] N,                    // 4-bit input
    output reg R
    );

always @ (*)
   if (N == 1) R = 0;                   // 1 is not prime
   else if ( (N % 2) == 0 )             // Divisible by 2?
      begin if (N == 2) R = 1;
                    else R = 0;
      end
   else if (N <= 7) R = 1;
   else if ( (N == 11) || (N == 13) ) R = 1;
   else R = 0;
endmodule
```

Example:
`PrimeDetector.v`
`PrimeDetector_Tester.v`