CENG 311
Computer Architecture

Lecture 2

Introduction to FPGA and VHDL

Asst. Prof. Tolga Ayav, Ph.D.

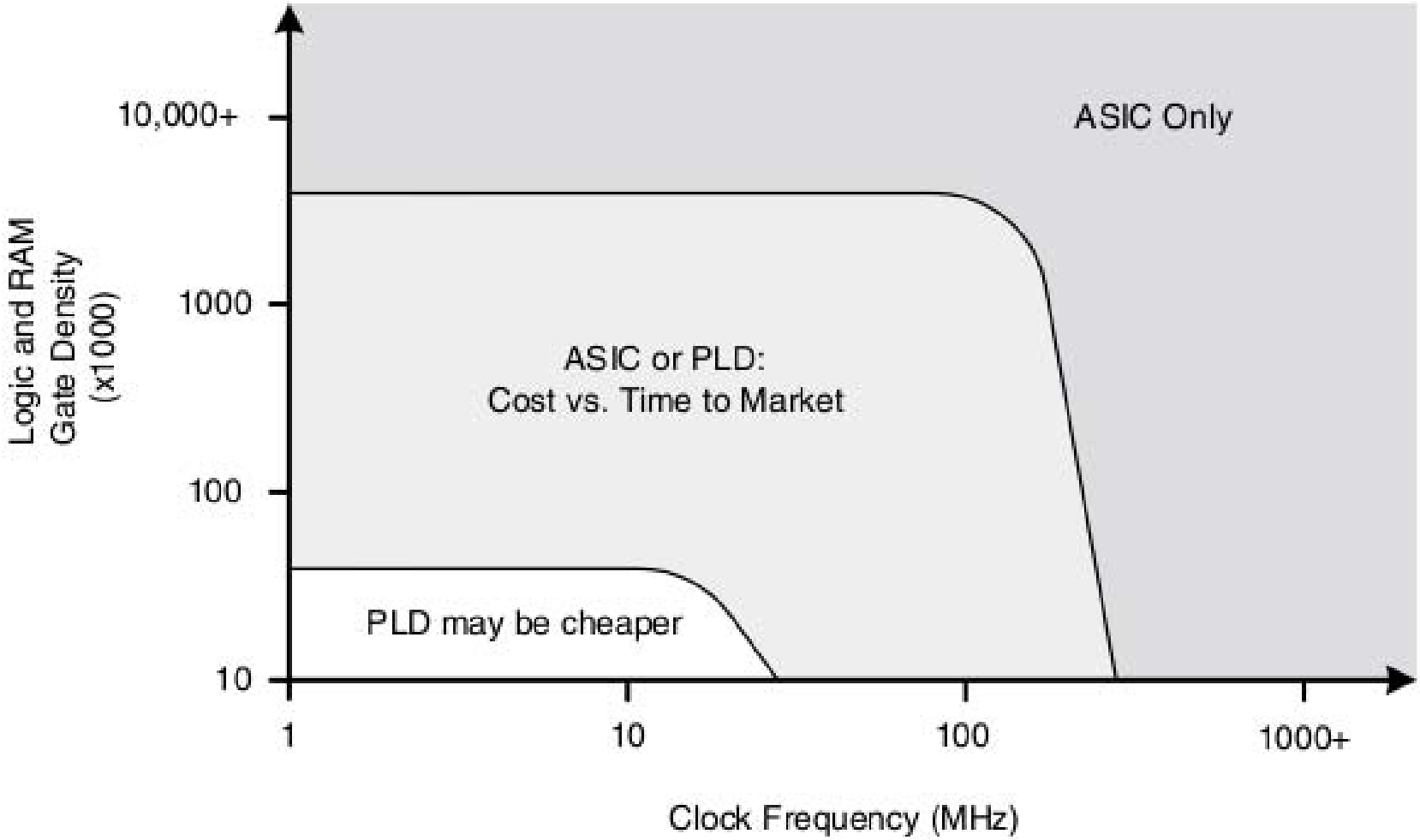
Department of Computer Engineering
İzmir Institute of Technology

Programmable Logic Devices (PLD)

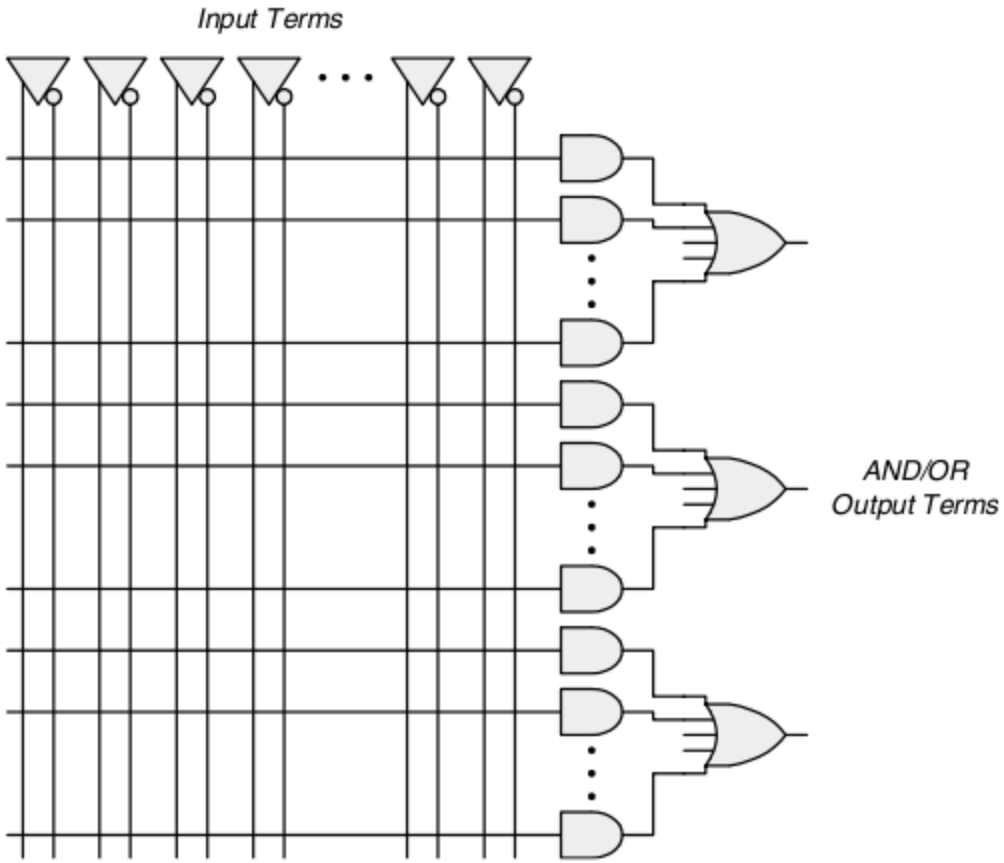
A programmable logic device or PLD is an electronic component used to build reconfigurable digital circuits. Unlike a logic gate, which has a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed (i. e. reconfigured).

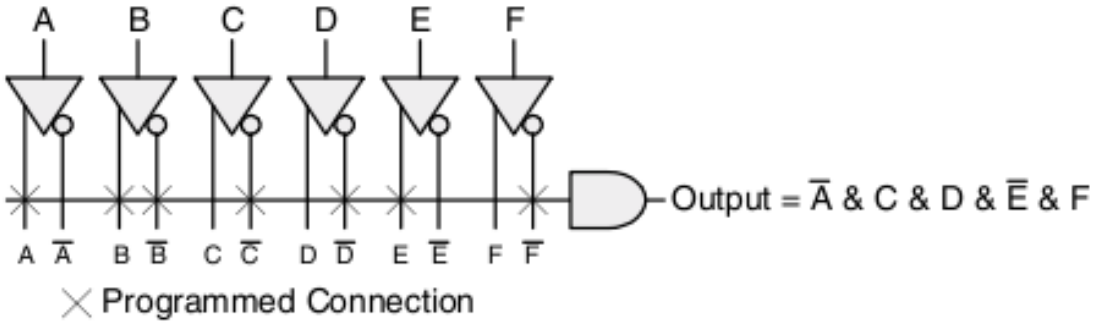
- ROM: Read Only Memory
- PAL: Programmable Array Logic
- GAL: Generic Array Logic
- CPLD: Complex Programmable Logic Device
- FPGA: Field Programmable Gate Array

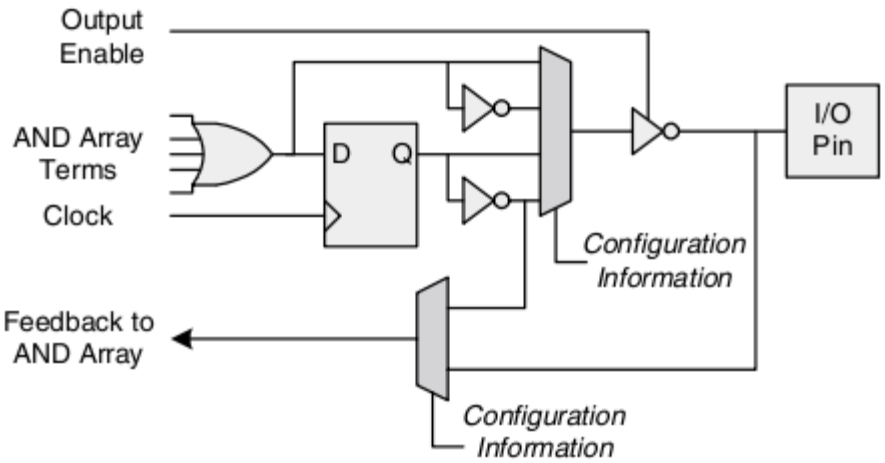
PLD vs. ASIC


ASIC Only
ASIC or PLD:
Cost vs. Time to Market
PLD may be cheaper
Logic and RAM
Gate Density
(x1000)
10,000+
1000
100
10
1
10
100
1000+
Clock Frequency (MHz)

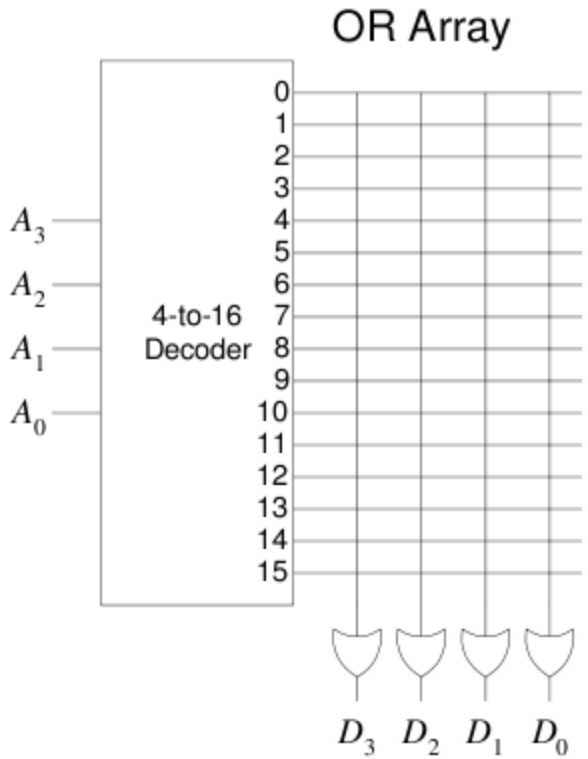
GALs and PALs

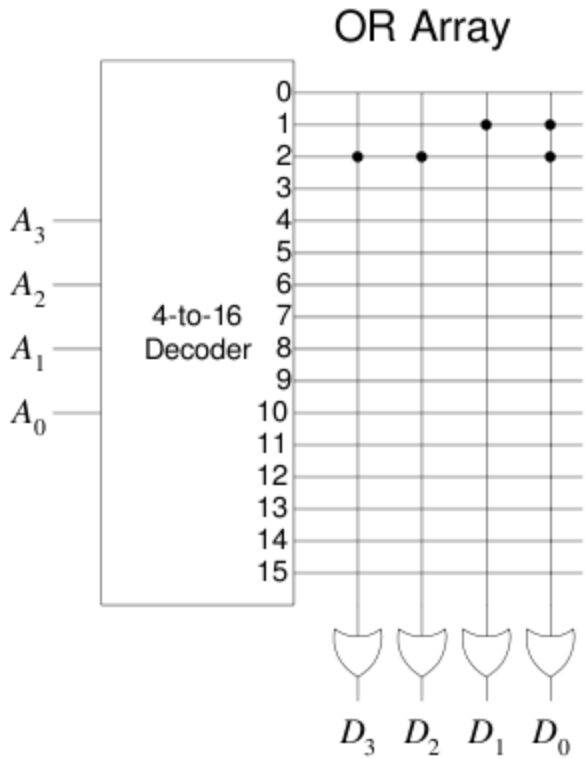



A
B
C
D
E
F
Output = $\overline{A}$ & C & D & $\overline{E}$ & F
A $\overline{A}$ B $\overline{B}$ C $\overline{C}$ D $\overline{D}$ E $\overline{E}$ F $\overline{F}$
Programmed Connection


Output Enable
AND Array Terms
Clock
D
Q
I/O Pin
Configuration Information
Feedback to AND Array
Configuration Information

Using ROMs to Implement a Function



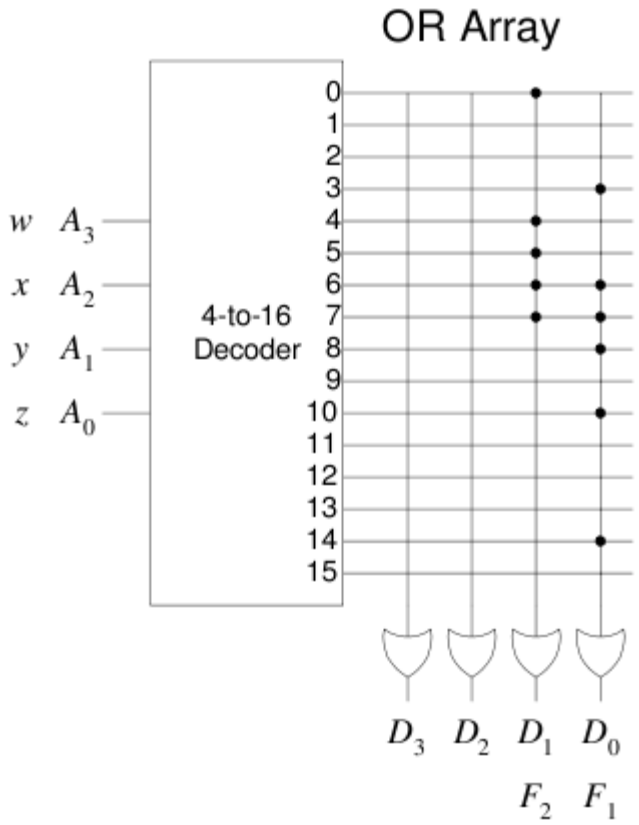


Example: Using 16x4 ROM to implement a function

Implement the following function:

$$F_1(w,x,y,z) = w'x'yz + w'xyz' + w'xyz + wx'y'z' + wx'yz' + wxyz'$$

$$F_2(w,x,y,z) = w'x'y'z' + w'x$$



Example: Using 4x4 PAL to implement a function

Implement the following function:

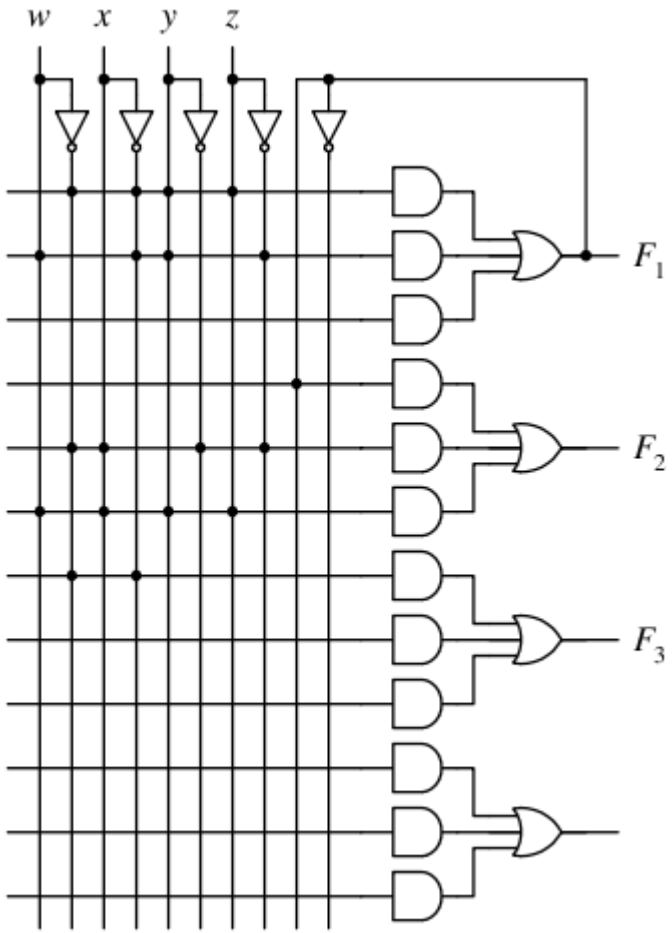
$$F_1(w,x,y,z) = w'x'yz + wx'yz'$$
$$F_2(w,x,y,z) = w'x'yz + wx'yz' + w'xy'z' + wxyz$$
$$F_3(w,x,y,z) = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz$$

First we can reduce the terms like as follows:

$$F_2(w,x,y,z) = w'x'yz + wx'yz' + w'xy'z' + wxyz$$
$$= F_1 + w'xy'z' + wxyz$$

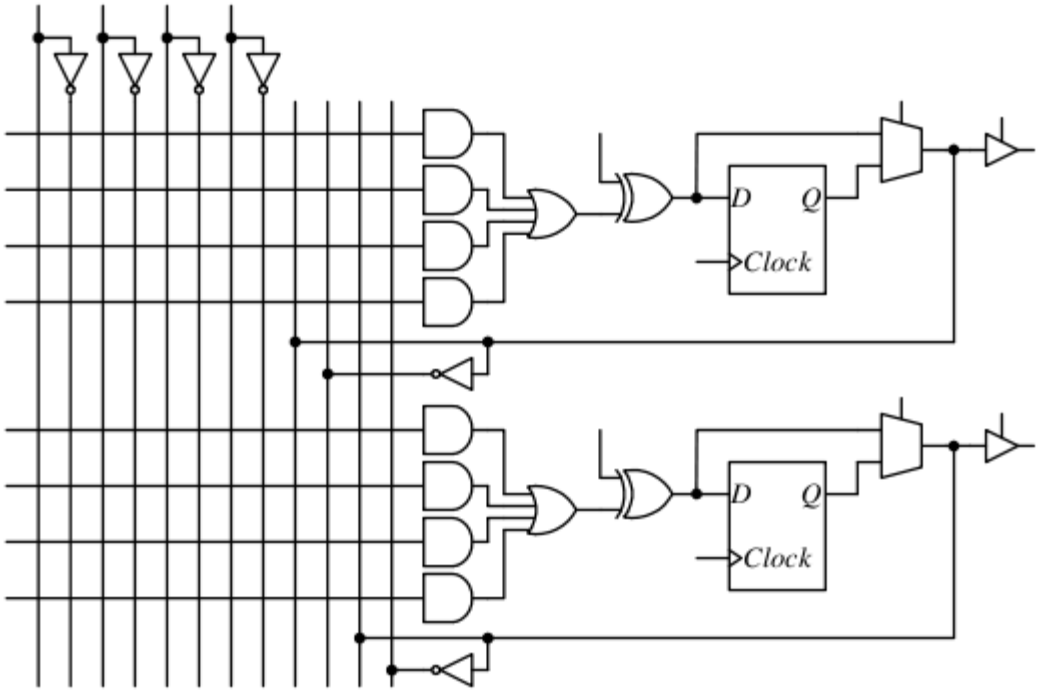
$$F_3(w,x,y,z) = w'x'y'z' + w'x'y'z + w'x'yz' + w'x'yz$$
$$= w'x'(y'z' + y'z + yz' + yz)$$
$$= w'x'$$

=>

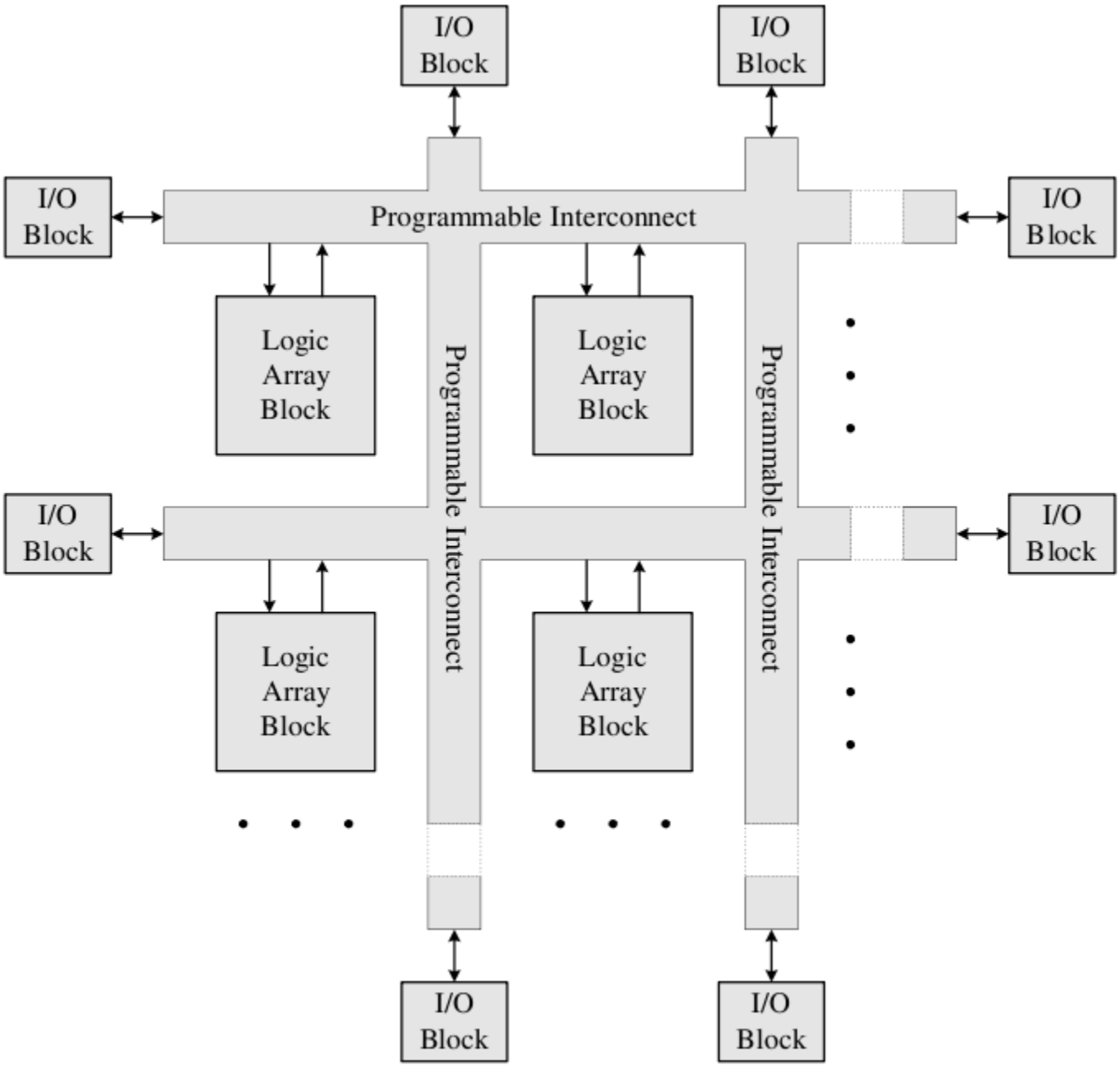


Complex Programmable Logic Device (CPLD)

(CPLD) is capable of implementing a circuit with upwards of 10,000 logic gates. Sequential circuits can also be implemented with CPLDs.

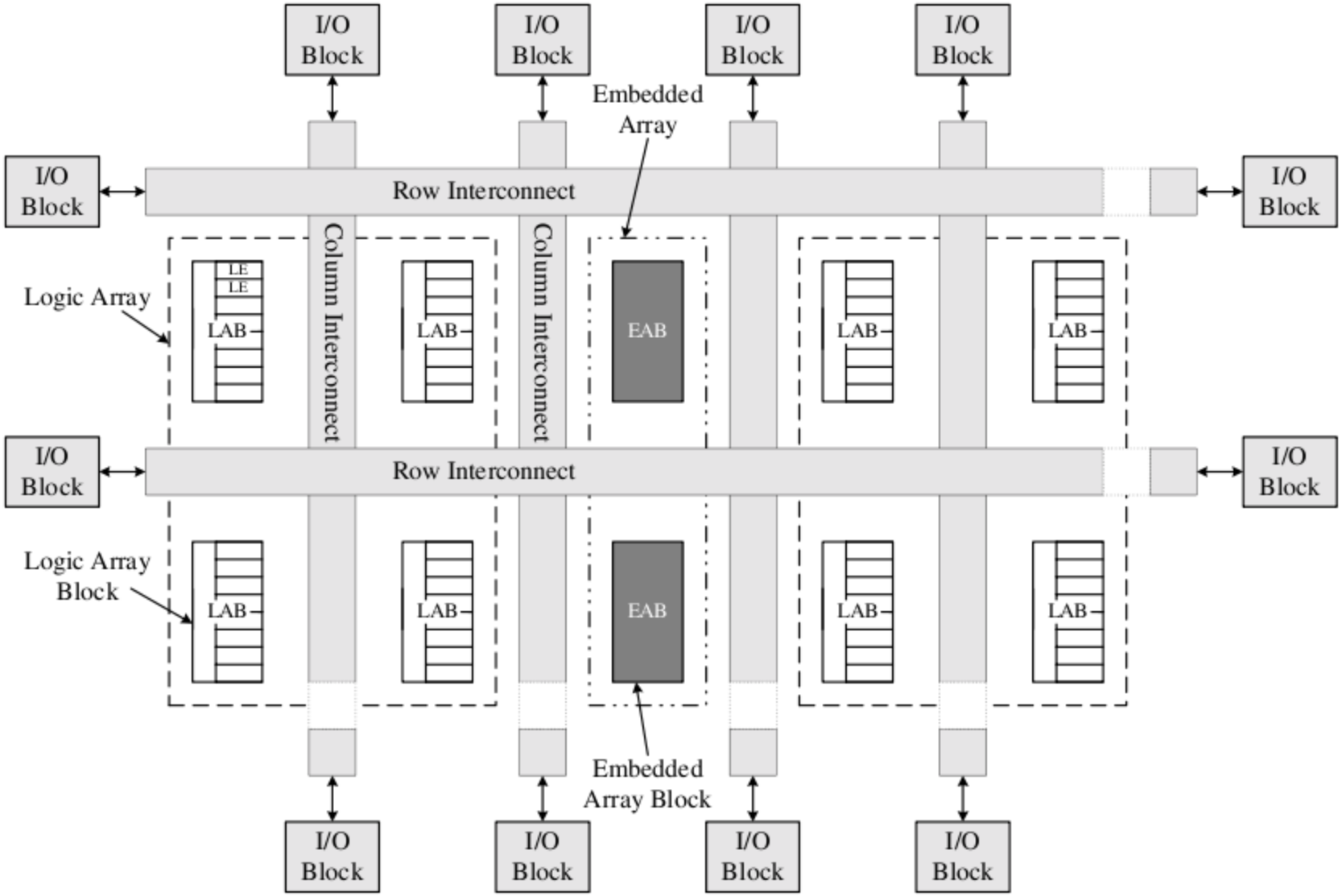


Internal Structure of CPLDs



Field Programmable Gate Array (FPGA)

Field programmable gate arrays (FPGAs) are complex programmable logic devices that are capable of implementing up to 250,000 logic gates and up to 40,960 RAM bits, as featured by the Altera FLEX10K250 FPGA chip.



FPGA Internal Structure (1)

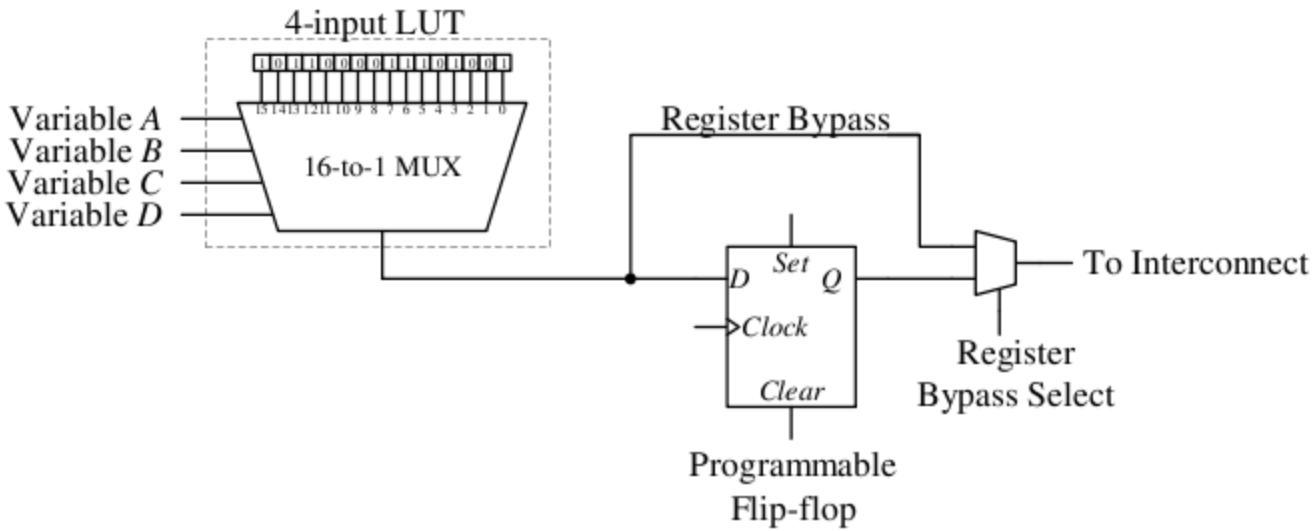
EAB:

The embedded array consists of a series of embedded array blocks (EAB). When implementing memory functions, each EAB provides 2,048 bits, which can be used to create RAM, dual-port RAM, or ROM. EABs can be used independently, or multiple EABs can be combined to implement larger functions.

LAB:

The logic array consists of multiple logic array blocks (LAB). Each LAB contains eight logic elements (LE) and a local interconnect. LE is the smallest logical unit in the FLEX10K architecture. Each LE consists of a 4-input look-up table (LUT) and a programmable flip-flop. The 4-input LUT is a function generator made from a 16-to-1 multiplexer that can quickly compute any function of four variables.

FPGA Internal Structure (2)

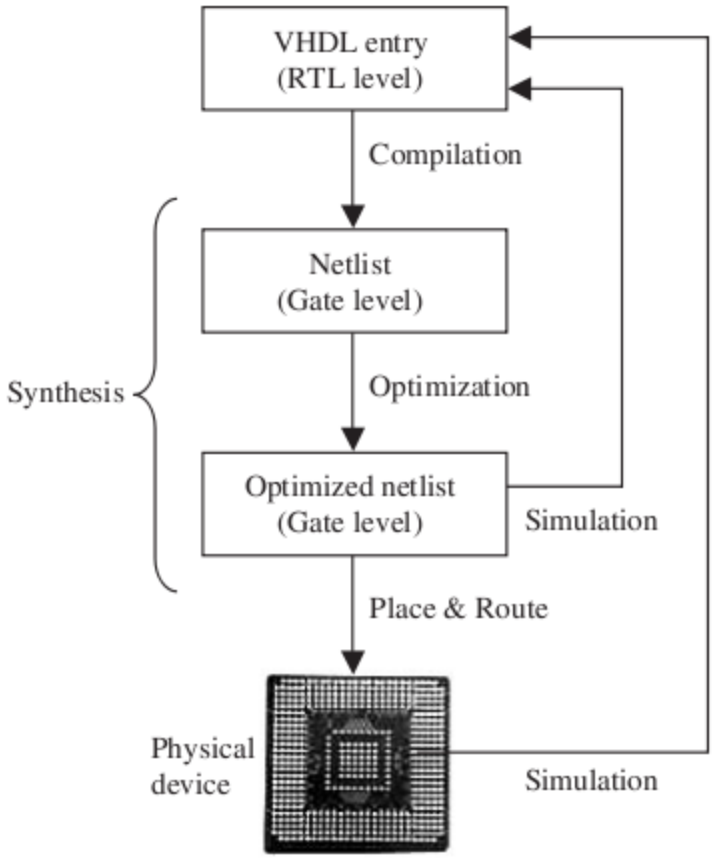


All the EABs, LABs, and I/O elements, are connected together via the FastTrack interconnect, which is a series of fast row and column buses that run the entire length and width of the device. The interconnect contains programmable switches so that the output of any block can be connected to the input of any other block.

Each I/O pin in an I/O element is connected to the end of each row and column of the interconnect and can be used as either an input, output, or bi-directional port.

VHDL

VHDL is a hardware description language. It describes the behavior of an electronic
circuit or system, from which the physical circuit or system can then be attained (implemented).



EDA (Electronic Design Automation) Tools

Altera's Quartus II

Xilinx's ISE suite

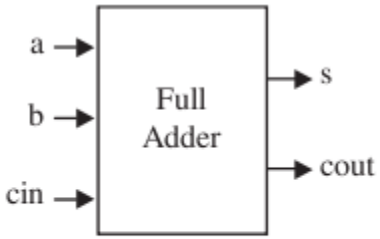
ModelSim (a simulator from Model Technology, a Mentor Graphics company)

Leonardo Spectrum (a synthesizer from Mentor Graphics)

Synplify (a synthesizer from Synplicity)

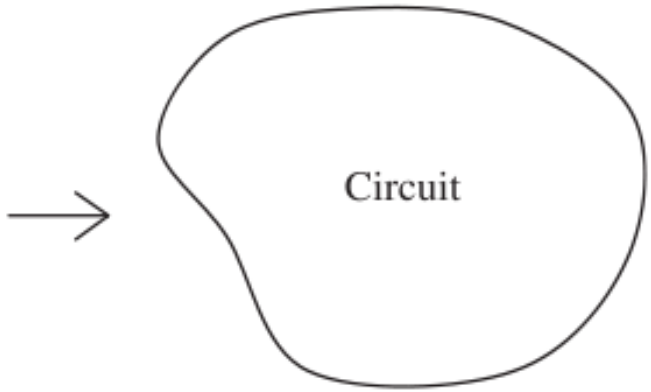
MaxPlus II

Translation of VHDL Code into a Circuit



a b cin	s cout
0 0 0	0 0
0 1 0	1 0
1 0 0	1 0
1 1 0	0 1
0 0 1	1 0
0 1 1	0 1
1 0 1	0 1
1 1 1	1 1

```
ENTITY full_adder IS
PORT (a, b, cin: IN BIT;
      s, cout: OUT BIT);
END full_adder;
---------------------------------------
ARCHITECTURE dataflow OF full_adder IS
BEGIN
   s <= a XOR b XOR cin;
   cout <= (a AND b) OR (a AND cin) OR
           (b AND cin);
END dataflow;
```



VHDL Code Structure

Fundamental VHDL Units:

LIBRARY declarations: Contains a list of all libraries to be used in the design. For example: ieee, std, work, etc.

ENTITY: Specifies the I/O pins of the circuit.

ARCHITECTURE: Contains the VHDL code proper, which describes how the circuit should behave (function).

Library declarations:

```
LIBRARY ieee;                   -- A semi-colon (;) indicates
USE ieee.std_logic_1164.all; -- the end of a statement or

LIBRARY std;                    -- declaration, while a double
USE std.standard.all;           -- dash (--) indicates a comment.

LIBRARY work;
USE work.all;
```

VHDL Syntax

$$
\begin{array}{lcll}
P & ::= & \texttt{entity } N_1 \texttt{ is port}(R) \texttt{ end } N_1; & \\
 & & \texttt{architecture } N_2 \textit{ of } N_1 \texttt{ is} & \\
 & & [D] \texttt{ begin } C \texttt{ end } N_2; & (\textit{Circuit declaration}) \\
C & ::= & s <= e & (\textit{Signal assignment}) \\
 & | & s <= e \texttt{ when } b & (\textit{Conditional signal assig.}) \\
 & | & \texttt{process}(W) \texttt{ is } [D] \texttt{ begin } S \texttt{ end} & (\textit{Process}) \\
 & | & \texttt{for } v \texttt{ in } i_1 \texttt{ to } i_2 \texttt{ generate } C & (\textit{Generate}) \\
 & | & \texttt{entity } N \texttt{ port map}(W) & (\textit{Comp. instantiation}) \\
 & | & C_1; C_2 & (\textit{Parallel composition}) \\
S & ::= & v{:=}e & (\textit{Variable assignment}) \\
 & | & s <= e & (\textit{Signal assignment}) \\
 & | & a(e_1){:=}e_2 & (\textit{Array assignment}) \\
 & | & \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2 \texttt{ endif} & (\textit{conditional}) \\
 & | & \texttt{case } e \texttt{ when } i_1 => S_1 \ldots & \\
 & & \texttt{ when } i_n => S_n \texttt{ end case} & (\textit{conditional}) \\
 & | & \texttt{for } v \texttt{ in } 0 \texttt{ to } i & \\
 & & \texttt{loop } S \texttt{ end loop} & (\textit{Iteration}) \\
 & | & S_1; S_2 & (\textit{sequencing})
\end{array}
$$

VHDL Syntax (cont'd)

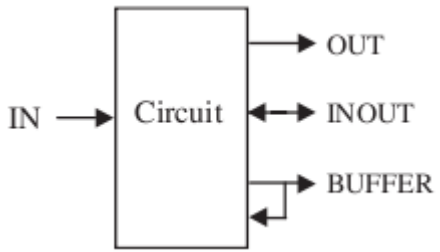
$$
\begin{array}{lcl}
b & ::= & b_1 \odot b_2 \mid \texttt{true} \mid \texttt{false} \\
 & \mid & v \mid s \mid i \mid \neg b \\
 & \mid & \texttt{rising\_edge(s)} \\
 & \mid & \texttt{falling\_edge(s)} \\
e & ::= & i \mid v \mid s \mid a(e) \mid e_1 \odot e_2 \\
 & \mid & e_1 + e_2 \mid e_1 * e_2 \\
D & ::= & \texttt{variable } v : \texttt{integer } [:= i]; \\
 & \mid & \texttt{signal } s : \texttt{std\_logic } [:= '1' \mid '0']; \\
 & \mid & \texttt{signal } s : \texttt{std\_logic\_vector} \\
 & & \qquad (i_1 \texttt{ to } i_2)[:= i_3]; \\
 & \mid & D_1; D_2 \\
R & ::= & \texttt{signal } s : \texttt{std\_logic}; \qquad (\textit{Port declaration}) \\
 & \mid & \texttt{signal } s : \texttt{std\_logic\_vector} \\
 & & \qquad (i_1 \texttt{ to } i_2); \\
 & \mid & R_1; R_2
\end{array}
$$

ENTITY:

An ENTITY is a list with specifications of all input and output pins (PORTS) of the circuit. Its syntax:

```
ENTITY entity_name IS
   PORT (
      port_name : signal_mode signal_type;
      port_name : signal_mode signal_type;
      ...);
END entity_name;
```

The mode of the signal can be IN, OUT, INOUT, or BUFFER. IN and OUT are truly unidirectional pins, while INOUT is bidirectional. BUFFER, on the other hand, is employed when the output signal must be used (read) internally.

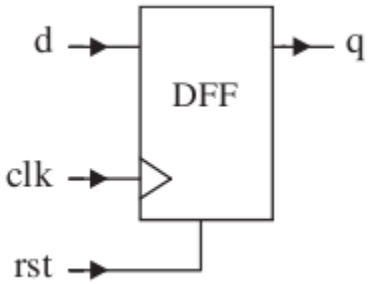


ARCHITECTURE:

The ARCHITECTURE is a description of how the circuit should behave (function). Its syntax is the following:

```
ARCHITECTURE architecture_name OF entity_name IS
    [declarations]
BEGIN
    (code)
END architecture_name;
```

Example: DFF with Asynchronous Reset




```
1  ------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY dff IS
6     PORT ( d, clk, rst: IN STD_LOGIC;
7            q: OUT STD_LOGIC);
8  END dff;
9  ------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12    PROCESS (rst, clk)
13    BEGIN
14       IF (rst='1') THEN
15          q <= '0';
16       ELSIF (clk'EVENT AND clk='1') THEN
17          q <= d;
18       END IF;
19    END PROCESS;
20 END behavior;
21 ------------------------------------
```

Data Types

Synthesizable data types.

Data types	Synthesizable values
BIT, BIT_VECTOR	'0', '1'
STD_LOGIC, STD_LOGIC_VECTOR	'X', '0', '1', 'Z' (resolved)
STD_ULOGIC, STD_ULOGIC_VECTOR	'X', '0', '1', 'Z' (unresolved)
BOOLEAN	True, False
NATURAL	From 0 to +2, 147, 483, 647
INTEGER	From −2,147,483,647 to +2,147,483,647
SIGNED	From −2,147,483,647 to +2,147,483,647
UNSIGNED	From 0 to +2,147,483,647
User-defined integer type	Subset of INTEGER
User-defined enumerated type	Collection enumerated by user
SUBTYPE	Subset of any type (pre- or user-defined)
ARRAY	Single-type collection of any type above
RECORD	Multiple-type collection of any types above

```
TYPE byte IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;              -- 1D
                                                           -- array
TYPE mem1 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;      -- 2D
                                                           -- array
TYPE mem2 IS ARRAY (0 TO 3) OF byte;                       -- 1Dx1D
                                                           -- array
TYPE mem3 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(0 TO 7);   -- 1Dx1D
                                                           -- array
SIGNAL a: STD_LOGIC;                                 -- scalar signal
SIGNAL b: BIT;                                       -- scalar signal
SIGNAL x: byte;                                      -- 1D signal
SIGNAL y: STD_LOGIC_VECTOR (7 DOWNTO 0);             -- 1D signal
SIGNAL v: BIT_VECTOR (3 DOWNTO 0);                   -- 1D signal
SIGNAL z: STD_LOGIC_VECTOR (x'HIGH DOWNTO 0);        -- 1D signal
SIGNAL w1: mem1;                                     -- 2D signal
SIGNAL w2: mem2;                                     -- 1Dx1D signal
SIGNAL w3: mem3;                                     -- 1Dx1D signal
```

Legal Scalar Assignments

```
x(2) <= a;            -- same types (STD_LOGIC), correct indexing
y(0) <= x(0);         -- same types (STD_LOGIC), correct indexing
z(7) <= x(5);         -- same types (STD_LOGIC), correct indexing
b <= v(3);            -- same types (BIT), correct indexing
w1(0,0) <= x(3);      -- same types (STD_LOGIC), correct indexing
w1(2,5) <= y(7);      -- same types (STD_LOGIC), correct indexing
w2(0)(0) <= x(2);     -- same types (STD_LOGIC), correct indexing
w2(2)(5) <= y(7);     -- same types (STD_LOGIC), correct indexing
w1(2,5) <= w2(3)(7);  -- same types (STD_LOGIC), correct indexing
```

Illegal Scalar Assignments

```
b <= a;                 -- type mismatch (BIT x STD_LOGIC)
w1(0)(2) <= x(2);       -- index of w1 must be 2D
w2(2,0) <= a;           -- index of w2 must be 1Dx1D
```

Legal Vector Assignments

```
x <= "11111110";
y <= ('1','1','1','1','1','1','0','Z');
z <= "11111" & "000";
x <= (OTHERS => '1');
y <= (7 =>'0', 1 =>'0', OTHERS => '1');
z <= y;
y(2 DOWNTO 0) <= z(6 DOWNTO 4);
w2(0)(7 DOWNTO 0) <= "11110000";
w3(2) <= y;
z <= w3(1);
z(5 DOWNTO 0) <= w3(1)(2 TO 7);
w3(1) <= "00000000";
w3(1) <= (OTHERS => '0');
w2 <= ((OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'),(OTHERS=>'0'));
w3 <= ("11111100", ('0','0','0','0','Z','Z','Z','Z',),
       (OTHERS=>'0'), (OTHERS=>'0'));
w1 <= ((OTHERS=>'Z'), "11110000" ,"11110000", (OTHERS=>'0'));
```

Illegal Array Assignments

```
x <= y;                                -- type mismatch
y(5 TO 7) <= z(6 DOWNTO 0);            -- wrong direction of y
w1 <= (OTHERS => '1');                 -- w1 is a 2D array
w1(0, 7 DOWNTO 0) <="11111111";        -- w1 is a 2D array
w2 <= (OTHERS => 'Z');                 -- w2 is a 1Dx1D array
w2(0, 7 DOWNTO 0) <= "11110000";       -- index should be 1Dx1D
```

Single Bit Versus Bit Vector

```
-----------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT;
         x: OUT BIT);
END and2;
-----------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------
```

```
-----------------------------------
ENTITY and2 IS
   PORT (a, b: IN BIT_VECTOR (0 TO 3);
         x: OUT BIT_VECTOR (0 TO 3));
END and2;
-----------------------------------
ARCHITECTURE and2 OF and2 IS
BEGIN
   x <= a AND b;
END and2;
-----------------------------------
```

Components

```
library ieee;
use ieee.std_logic_1164.all;
entity add2 is
  port (
    A, B : in std_logic_vector(1 downto 0);
    C    : out std_logic_vector(2 downto 0));
end add2;
architecture imp of add2 is
  component full_adder
    port (
      a, b, c     : in std_ulogic;
      sum, carry : out std_ulogic);
  end component;
  signal carry : std_ulogic;

begin
  bit0 : full_adder port map (
    a      => A(0),
    b      => B(0),
    c      => '0',
    sum    => C(0),
    carry => carry);
  bit1 : full_adder port map (
    a      => A(1),
    b      => B(1),
    c      => carry,
    sum    => C(1),
    carry => C(2));
end imp;
```

Multiplexer (using when...else)

```
library ieee;
use ieee.std_logic_1164.all;

entity multiplexer_4_1 is
  port(in0, in1, in2, in3 : in
 std_ulogic_vector(15 downto 0);
       s0, s1             : in std_ulogic;
       z                  : out
 std_ulogic_vector(15 downto 0));
end multiplexer_4_1;

architecture imp of multiplexer_4_1 is
begin
  z <= in0 when (s0 = '0' and s1 = '0')
       in1 when (s0 = '1' and s1 = '0')
       in2 when (s0 = '0' and s1 = '1')
       in3 when (s0 = '1' and s1 = '1')
       "XXXXXXXXXXXXXXX";
end imp;
```

Multiplexer (using with...select)

```
library ieee;
use ieee.std_logic_1164.all;

entity multiplexer_4_1 is
  port(in0, in1, in2, in3 : in
 std_ulogic_vector(15 downto 0);
       s0, s1             : in std_ulogic;
       z                  : out
 std_ulogic_vector(15 downto 0));
end multiplexer_4_1;

architecture usewith of multiplexer_4_1 is
  signal sels : std_ulogic_vector(1 downto 0); -- Local wires
begin
  sels <= s1 & s0;                             -- vector
    concatenation
  with sels select
    z <=
    in0                when "00",
    in1                when "01",
    in2                when "10",
    in3                when "11",
    "XXXXXXXXXXXXXXXX" when others;
end usewith;
```

Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity dec1_8 is
port (
  sel : in std_logic_vector(2 downto 0);
  res : out std_logic_vector(7 downto 0));
end dec1_8;


architecture imp of dec1_8 is
begin
  res <= "00000001" when sel = "000" else
         "00000010" when sel = "001" else
         "00000100" when sel = "010" else
         "00001000" when sel = "011" else
         "00010000" when sel = "100" else
         "00100000" when sel = "101" else
         "01000000" when sel = "110" else
         "10000000";
end imp;
```

A Very Primitive ALU

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity alu is
  port (
    A, B : in std_logic_vector(7 downto 0);
    ADD :  in std_logic;
    RES :  out std_logic_vector(7 downto 0));
end alu;


architecture imp of alu is
begin
  RES <= A + B when ADD = '1' else
         A - B;
end imp;
```

Homework

1. Download Modelsim (Ask the instructor for the download location.)

2. Write down and Save the VHDL code for “A primitive ALU” in alu1.vhd file.

3. Compile and then try to simulate the circuit!