School on DM dark matter & Nu neutrino 2018

School and Workshop on Dark Matter and Neutrino Detection
July 23 - August 3, 2018
São Paulo, Brazil
ICTP-SAIFR/IFT-UNESP

High-speed data acquisition and optimal filtering based on programmable logic for single-photoelectron (SPE) measurement setup

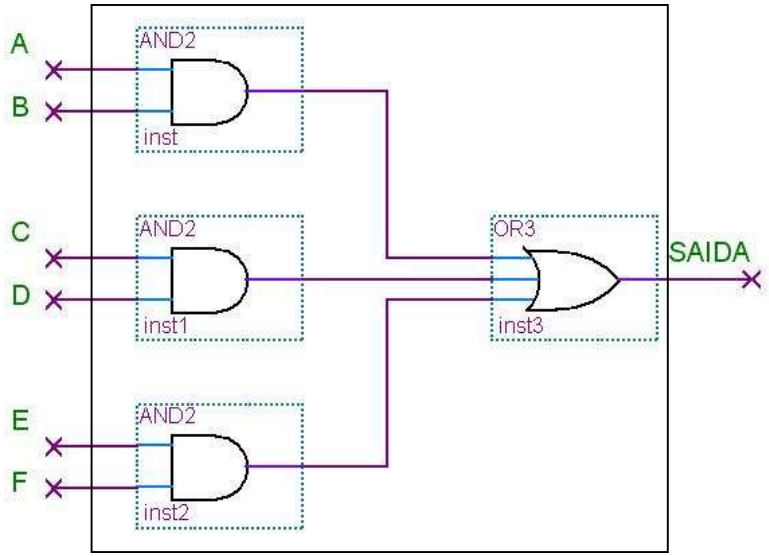
Experiment #7

Herman P. Lima Jr (CBPF), Rafael Nobrega (UFJF)
hlima@cbpf.br, rnobrega@gmail.com

ICTP SAIFR | International Centre for Theoretical Physics
South American Institute for Fundamental Research

Campus of IFT-UNESP - São Paulo, Brasil

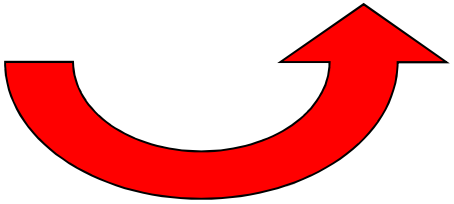
Challenge




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

entity logica is
   port (A,B,C : in  std_logic;
         D,E,F : in  std_logic;
         SAIDA : out std_logic);
end logica;

architecture v_1 of logica is
begin
   SAIDA <= (A and B) or (C and D) or (E and F);
end v_1;
```



References

- **Fundamentals of Digital Logic with VHDL Design**, *Stephen Brown, Zvonko Vranesic,* McGraw-Hill, 2000.
- **The Designer's Guide to VHDL**, *Peter Ashenden*, 2nd Edition, Morgan Kaufmann, 2002.
- **VHDL Coding Styles and Methodologies**, *Ben Cohen*, 2nd Edition, Kluwer Academic Publishers, 1999.
- **Digital Systems Design with VHDL and Synthesis: An Integrated Approach**, *K. C. Chang*, Wiley-IEEE Computer Society Press, 1999.
- **Application-Specific Integrated Circuits**, *Michael Smith*, Addison-Wesley, 1997.

- www.altera.com (*datasheets, application notes, reference designs*)
- www.xilinx.com (*datasheets, application notes, reference designs*)
- www.doulos.com/knowhow/vhdl_designers_guide (*The Designer's Guide to VHDL*)
- www.acc-eda.com/vhdlref/index.html (*VHDL Language Guide*)
- www.vhdl.org

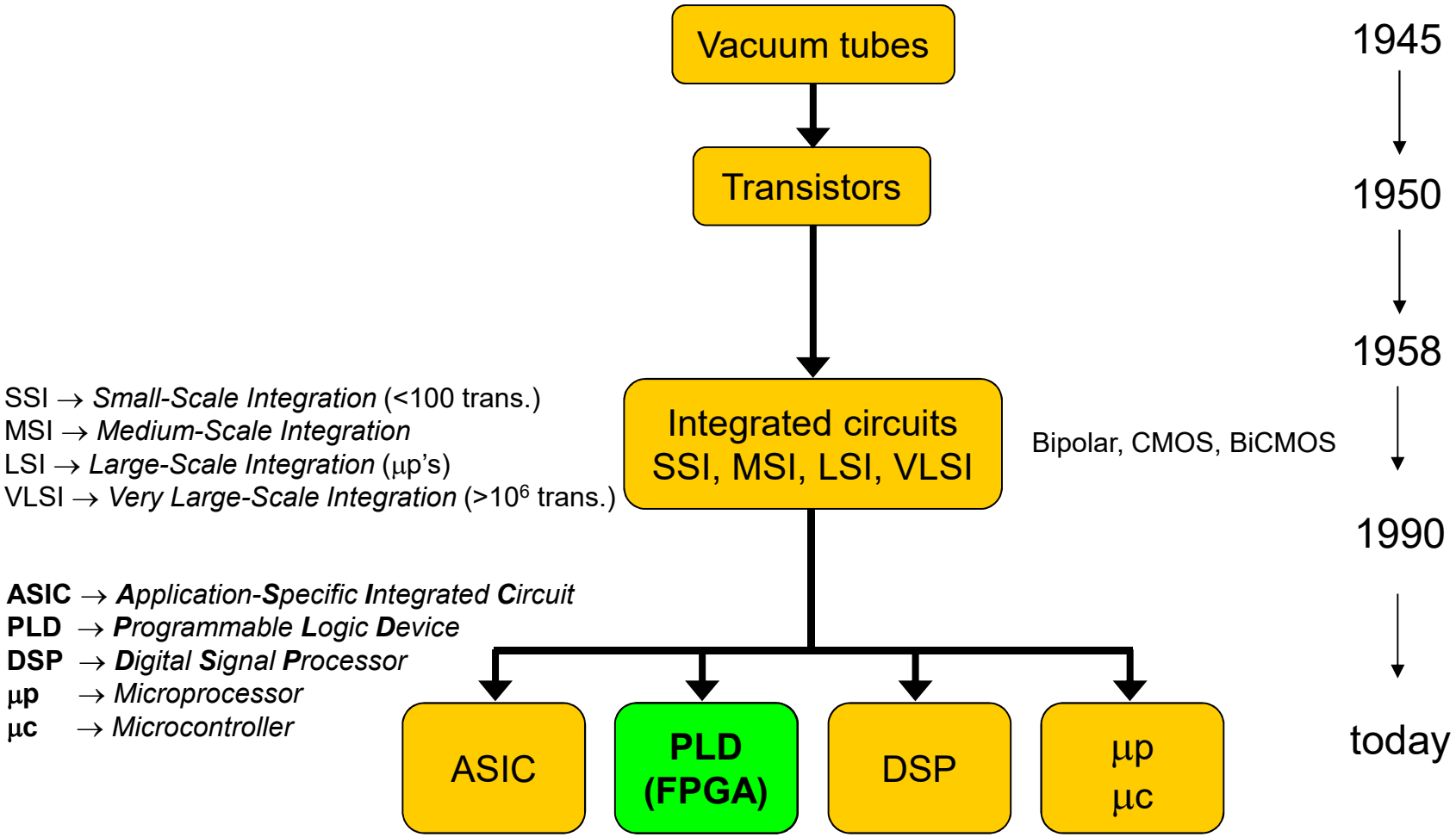
Background required

- Digital Electronics:
  - ✓ logic gates
  - ✓ flip-flops
  - ✓ multiplexers
  - ✓ comparators
  - ✓ counters
  - ✓ ...

Agenda

- Digital electronics: evolution, current technologies
- Programmable Logic
- Introduction to VHDL (for synthesis)

Digital Electronics: evolution



Digital Electronics: current technologies

	ASIC	PLD	DSP	Controllers, processors
Definition	Specific integrated circuits (specific applications)	CPLD, FPGA (user-programmable applications)	High-performance math processors	Microcontrollers, computer processors
Reconfigurable ?	**No**	**Yes (hardware)**	**Yes (software)**	**Yes (software)**
Basic features	Very high speed Very high density Low power	High speed High density Medium/high power	High speed, Medium power	High speed, Low power
Unit cost ($)	↑ qty ⇒ low ↓ qty ⇒ prohibitive	↑ qty ⇒ high ↓ qty ⇒ low	↑ qty ⇒ medium ↓ qty ⇒ medium	↑ qty ⇒ low ↓ qty ⇒ low
Design entry level	Low / Medium (schematics / code)	Low / Medium (schematics / code)	High (code)	Medium / High (code)

Programmable Logic

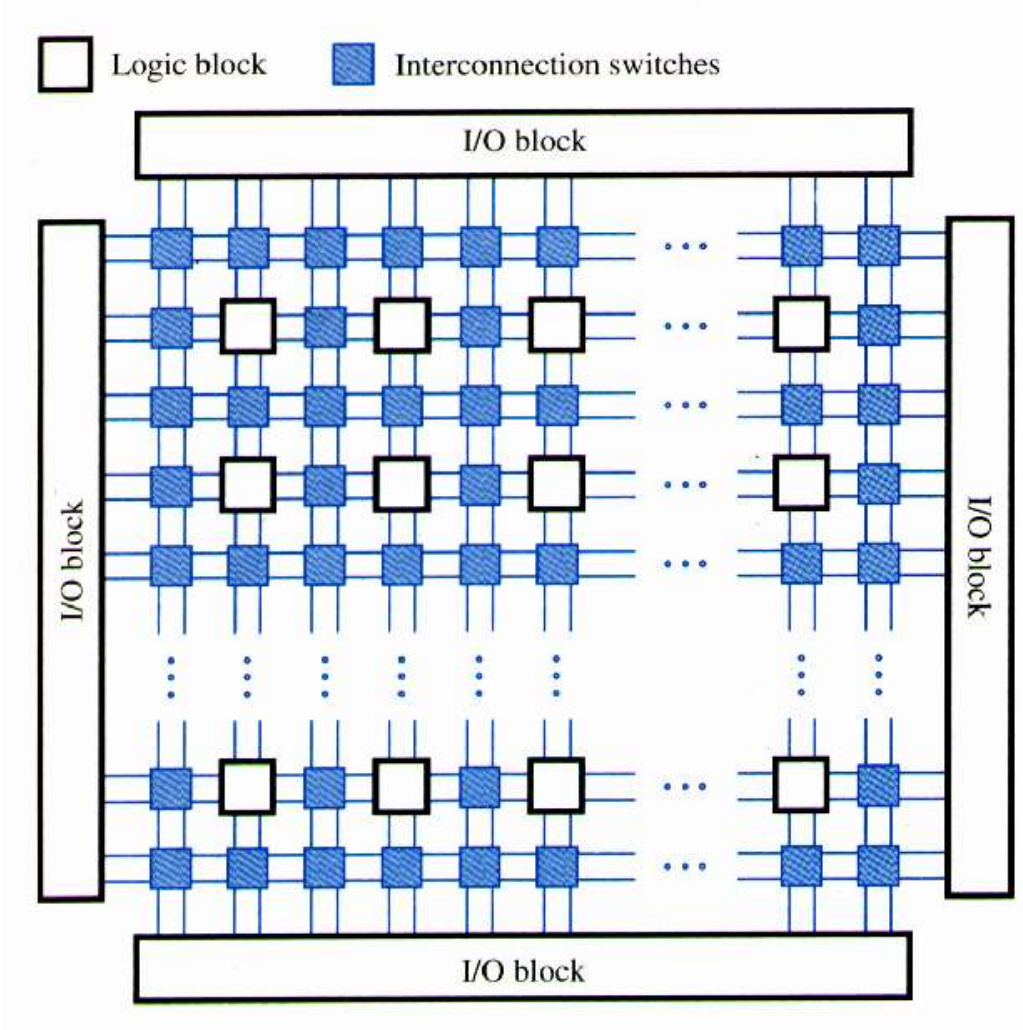
Field Programmable Gate Array

- What is it? ⇒ semiconductor device with programmable logic
- Applications ⇒ any system that requires digital circuits of medium to high complexity (high-speed, high density, segmented memory blocks).
- Features
  - ✓ reprogrammable, in practice, for an indefinite number of times
  - ✓ configuration technologies: SRAM, Antifuse, Flash
  - ✓ external memory needed (EEPROM, flash) for the design (SRAM technology)
  - ✓ portable languages for any tools (VHDL, Verilog)
  - ✓ high density of programmable logic
  - ✓ high density of flip-flops (ideal for synchronous designs)
  - ✓ rich libraries of basic blocks (multiplexers, decoders, ...)
  - ✓ dedicated blocks (DSP, memory, processors, PLL, SERDES, ...)
  - ✓ several electrical standards for interfacing (LVTTL, LVCMOS, LVDS, ...)
  - ✓ programmable through *IP Cores* (ex: communication interfaces (PCIe))
  - ✓ migration FPGA → ASIC (ex: *Hardcopy Series*, from Intel)

Programmable Logic

FPGA structure

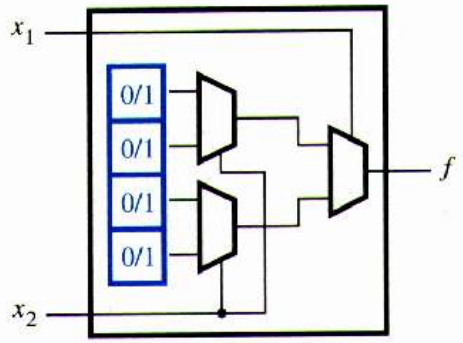
- Matrix of logic blocks.
- Horizontal and vertical connection channels.
- **Logic Block:** location of the available logic elements.
- **I/O blocks:** communication with the external circuits.
- **Interconnection switches:** connection inbetween logic blocks and between logic blocks and I/O pins.



Programmable Logic

Logic Blocks

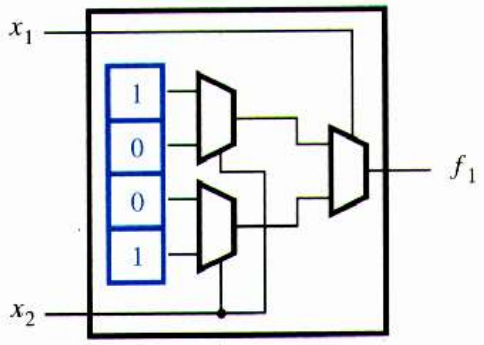
- *Look-Up Table* (LUT): cells with memory and multiplexers.
- LUTs are used to implement a logic function.
- Number of memory cells equal to $2^{(\text{number of inputs})}$.
- Implementation is transparent to the designer.
- Memory cells are volatile.



(a) Circuit for a two-input LUT

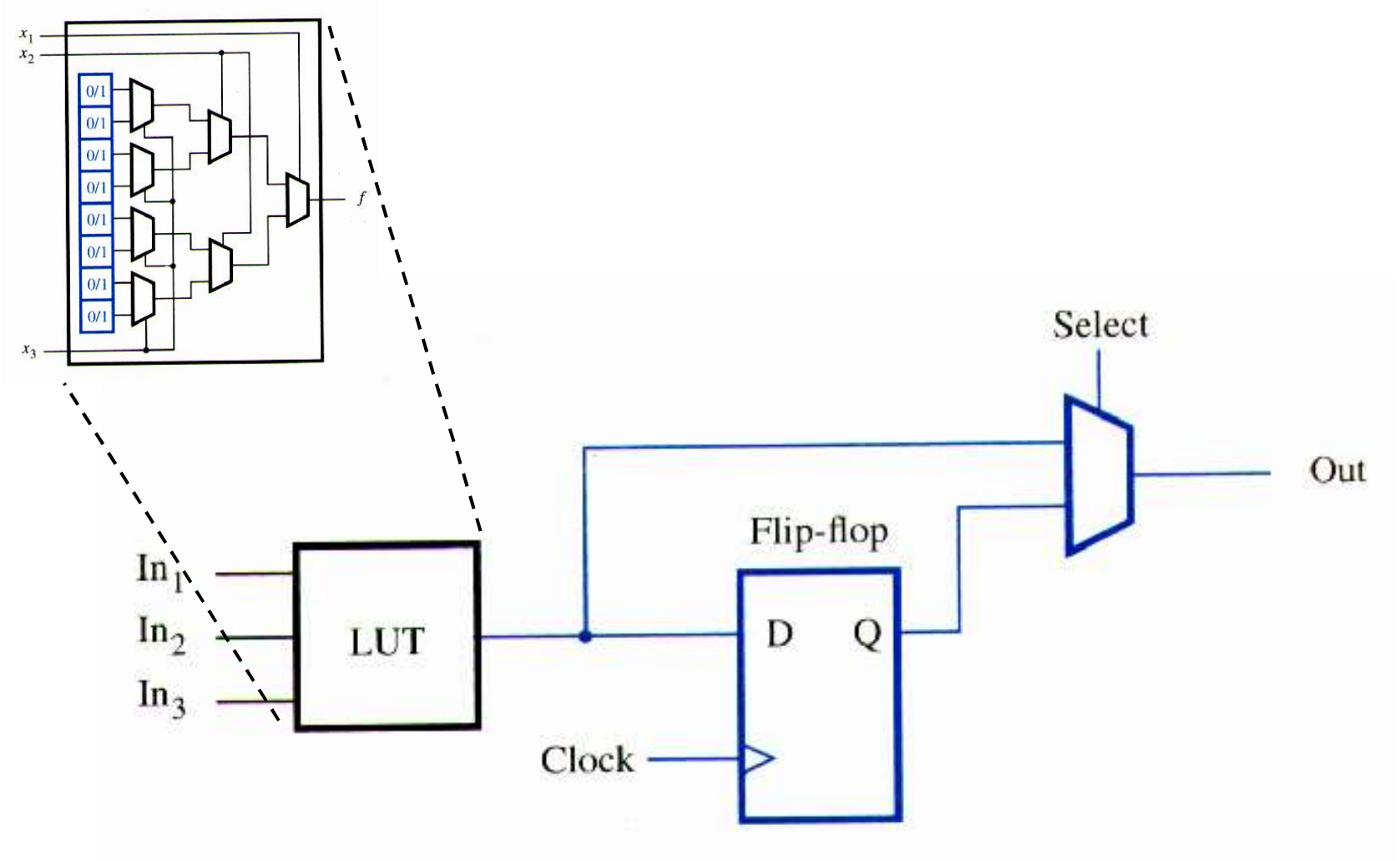
x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

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



(c) Storage cell contents in the LUT

Programmable Logic

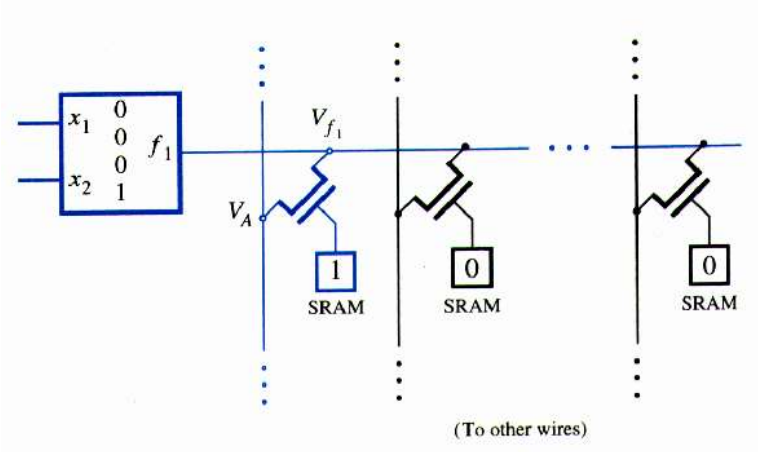


Example of *logic block* composed of a 3-inputs LUT and a register (D flip-flop).

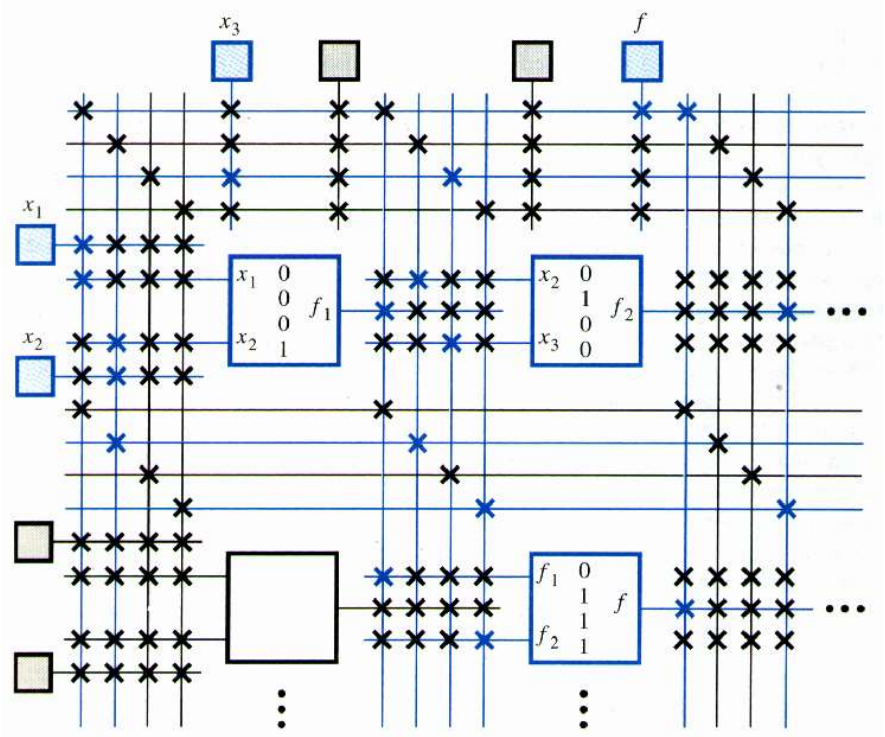
Programmable Logic

Example of design

- LUT with 2 inputs.
- 4 wires of interconnnection.
- Blue cells are activated.
- $f = f_1 + f_2 = x_1x_2 + x_2x_3$
- Interconnection switches are programmed by SRAM memory cells.



Configuration scheme of the interconnection wires.



Example of implementation of a combinational function.

Programmable Logic

Comparison of a low cost FPGA and a high performance FPGA:

	Cyclone IV family (1) (16 options)	**Stratix IV family** (1) (17 options)
Technology	L=60 nm (1,2V or 1,0V *core voltage*)	L=40 nm (0,9V *core voltage*)
Logic Elements	6.272 - 149.760	72.600 - 813.050
Clock control	2 - 8 PLLs	3 - 12 PLLs
Memory blocks	270 – 6.480 kbits	6462 – 20.736 kbits
Multipliers	15 - 360 (18 bits X 18 bits)	384 - 1288 (18 bits X 18 bits)
Transceivers	2 - 8 (3,125 Gbps)	8 - 48 (11,3 Gbps)
I/O pins (max)	72 – 528	289 - 976
Programmable I/O	yes	yes
Unit Price (US$)	from US$12 up to US$645	from US$800 up to US$24.270

(1) *www.altera.com*

Introduction to VHDL

- Brief history
- Important standards for synthesis
- Objects in VHDL
- Structural description
- Functional description
- Interface (*entity*)
- Implementation (*architecture*)
- Packages
- Components
- Concurrent and Sequential Assignments
- Processes
- Keywords
- Simulation

Introduction to VHDL

- Modern digital systems may be too complex to be described only by schematics.
- In early 80's there is a need of another method to describe very complex integrated circuits. The outcome is the creation of the *Hardware Description Languages* (HDL's).
- Most popular languages: **VHDL** (Europe) e **Verilog** (USA).
- Languages featuring even higher level of abstraction are already available in order to modeling and verify complex digital systems (ex: *SystemVerilog* e *SystemC*).

Introduction to VHDL

	VHDL 1993	Verilog 1995	SystemC	Verilog 2001	System Verilog 3.1	Verilog 2005	VHDL 200X
Switch-level modeling	X	X		X	X	X	X
ASIC timing	X	X		X	X	X	X
Concurency	X	X	X	X	X	X	X
Design modularization	X	X	X	X	X	X	X
Gate-level modeling	X	X	X	X	X	X	X
Gate-level timing	X	X	X	X	X	X	X
Four-state logic	X	X	X	X	X	X	X
Event handling	X	X	X	X	X	X	X
Basic data types	X	X	X	X	X	X	X
Basic behavioral constructs	X	X	X	X	X	X	X
Dynamic generation of hardware	X				X	X	X
Configurations	X				X		X
Simple assertions	X				X	X	X
Assertions (formal methods)					X	X	X
Dynamic-memory allocation	X		X		X	X	X
Pointers	X		X		X		X
Multidimensional arrays	X		X	X	X	X	X
Records	X		X		X	X	X
Enumeration	X		X		X		X
Automatic variables	X		X	X	X	X	X
Signed numbers	X		X	X	X	X	X
User-defined logic types	X						X
User-defined resolution functions	X						X
Void type			X		X		
Union	X		X		X		X
Behavioral constructs	X		X		X	X	X
Classes with methods and inheritance			X		X		X
Sequential regular expressions	X	X	X	X	X	X	X
Temporal-property definitions					X	X	X
Scheduling for testbench and assertions					X	X	X
Semaphores	X				X		X
Stimulus generation					X	X	X
Constrained random data generator					X	X	
Coverage monitoring					X	X	X
Strings and strings operations	X				X		X
Standard C interface					X	X	X
Transaction modeling			X			X	X
Module encryption						X	

Introduction to VHDL

- VHDL is a language to describe digital electronic circuits of medium to high complexity. Not recommended for simple designs.

- The name stantds for: *VHSIC Hardware Description Language*, with VHSIC meaning *Very High Speed Integrated Circuit*.

- Created from a north american project due to the demand of a new standard language to describe the structure and funcionality of very complex integrated circuits.

- Adopted and standardized by the *Institute of Electrical and Electronics Engineers* (IEEE).

Introduction to VHDL

- Important features:
  - Structural description, that is, a design is composed of sub-designs and these ones are interconnected.
  - Specification of functions using similar statements used in standard programming languages (if, for, while).
  - Complete simulation of a design prior to the manufacturing of an integrated circuit (ASIC), or the configuration of a programmable logic device (FPGA).
- Advantage over schematics:
  - Better readability of a design. Possibility of partitioning a design more easily, decoupling its blocks.
  - Use of parameters that modify the funcionality of performance of a design.
  - Cost reduction on prototyping.
  - Time reduction to insert a new product in the market.
- VHDL $\Rightarrow$ Modeling - Simulation - **Synthesis**

Introduction to VHDL

Important standards for synthesis in VHDL

- **IEEE 1076-1993**

  Define the base (core) of the language for modeling, simulation and synthesis.

- **IEEE 1076.6-1999**

  Define the subset specific for synthesis (*Register Transfer Level* - RTL).

- **IEEE 1164-1993** (STD_LOGIC_1164)

  Define a multi-value logic system for signals:

  ‘U’ → *Unitialized*
  ‘X’ → *Forcing Unknown*
  ‘0’ → *Forcing 0*
  ‘1’ → *Forcing 1*
  ‘Z’ → *High Impedance*
  ‘W’ → *Weak Unknown*
  ‘L’ → *Weak 0*
  ‘H’ → *Weak 1*
  ‘-’ → *Don´t Care*

- **IEEE 1076.3** (*Numeric Standard*)

  Define the numeric types **signed** and **unsigned**, and the respective arithmetic and conversion functions for use with synthesis tools.

Introduction to VHDL

Objects in VHDL

- There are 3 types of objects: **signals**, **constants** and **variables**.
- The name of an object can make use of any alphanumeric caracter, obeying the following rules: (1) it cannot be a VHDL keyword, (2) it must begin with a letter, (3) it cannot finish with *underscore* (_), and (4) it cannot present two *underscore* caracters together.
- For synthesis, signals (*signal* keyword) are the most important, since they represent the communication wires between blocks of the design.
- There are 3 places where a signal may be declared: in the **entity**, in the declaration part of the **architecture**, and in the declaration part of a **package**.
- Declaration of a signal: **signal** <signal_name> : [type] ;
- The signal type defines its possible values and manipulation.

Introduction to VHDL

Common types of objects in VHDL

- **bit** and **bit_vector**
  - defined in the standards IEEE 1076 and IEEE 1164
  - **bit** type may assume values ‘0’ or ‘1’
  - **bit_vector** type is simply a linear *array* of **bit** objects
  - ex: `signal c: bit_vector (1 to 4);` `c(1) <= ‘1’;` `c <= “1010”;`

- **std_logic** and **std_logic_vector**
  - defined in the standard IEEE 1164
  - in order do use them, it must be included the following lines of code:

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

  - provide greater flexibility than **bit** types, allowing the following values: ‘0’, ‘1’, ‘Z’, ‘-’, ‘L’, ‘H’, ‘U’, ‘X’ ou ‘W’
  - values ‘0’, ‘1’, ‘Z’ and ‘X’ are the most useful for simulation and synthesis

Introduction to VHDL

Common types of objects in VHDL (cont.)

- **signed** and **unsigned**
  - defined in the packages numeric_std and std_logic_arith
  - the packages also define the implementation of the arithmetic operators (+, -, *)
  - they are similar to the type std_logic_vector, *arrays* of std_logic
  - their use help to clearly indicate in the code what representation is being used for the data (signed or unsigned)

- **integer**
  - defined for use with arithmetic operators (IEEE 1076)
  - the number of bits is not specified in the code, like a std_logic_vector object
  - by definition, an integer makes use of 32 bits, allowing values from $-(2^{31}-1)$ to $2^{31}-1$
  - integers may use less than 32 bits through the use of the range keyword

    signal x : integer **range** -127 to 127;

Introduction to VHDL

Common types of objects in VHDL (cont.)

- **boolean**
  - may assume the logic values TRUE or FALSE, corresponding to ‘1’ and ‘0’;
  - ex: `signal flag : boolean ;`

- Enumeration type
  - type defined by the designer;
  - very useful for defining the name of the states in Finite State Machines;
  - ex:

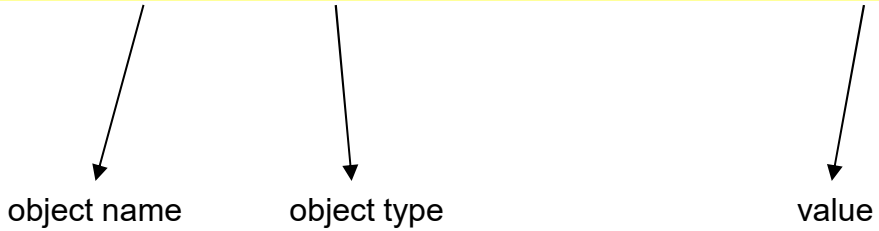
```
type state_id is (initiate, process);
signal y : state_id ;
 ...
y <= process;
```

Introduction to VHDL

Constants in VHDL

constant

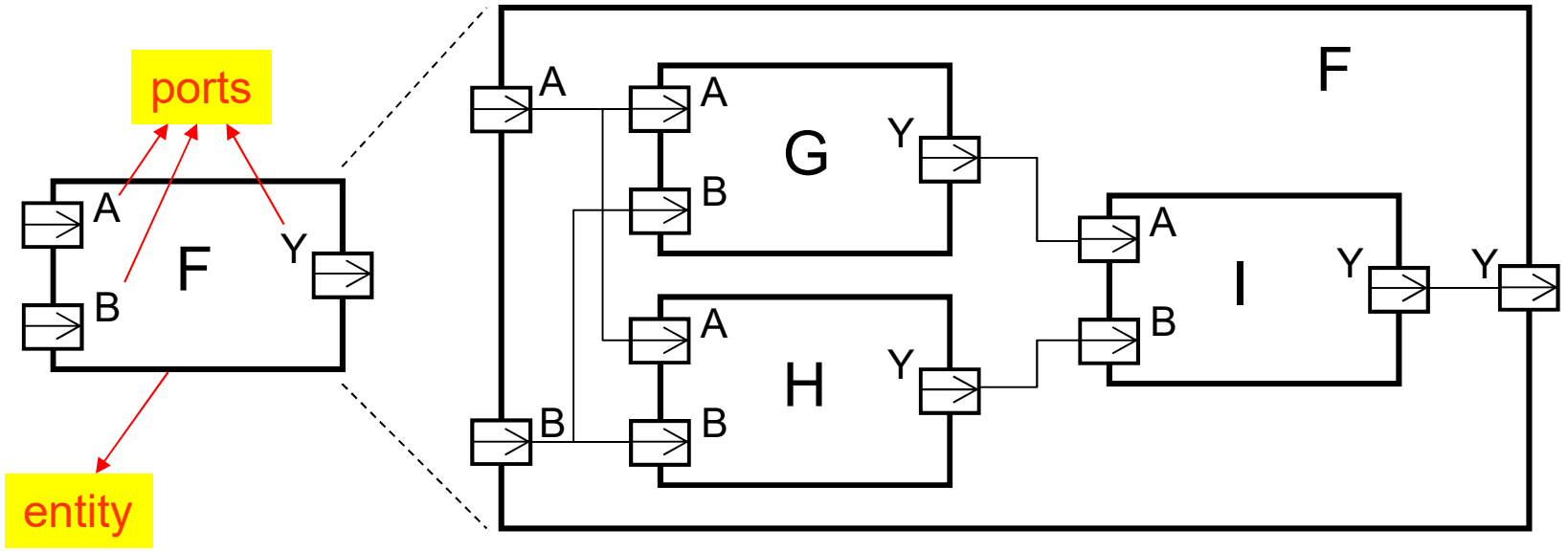
- an object of type constant cannot change its value throughout the code;
- a constant object may be defined without a value only inside packages;
- useful for improving the readability of the code;
- ex: **constant** z : std_logic_vector(2 **downto** 0) := “011”;



Introduction to VHDL

STRUCTURAL description:

- An electronic system may be described as a module with inputs and outputs.
- The output values may be function of:
  - only the current input values (**combinational circuit**)
  - current input values and internal states (**sequential circuit**)



Introduction to VHDL

FUNCTIONAL description:

- The electronic system is simply described by its function (ex: Y = ($\overline{A}$ and B) or (A and $\overline{B}$).
- Sequential systems obviously cannot be described only as a function of the inputs.

A VHDL design may be based on interconnected components in a complex hierarchy. Each component has an interface (entity) and an implementation (architecture), as follows:

component (full design)

interface

```
entity <entity_name> is
    port (a,b: in bit;
          y: out bit);
end <entity_name>;
```

implementation

```
architecture <arq_name> of < entity_name > is
begin
    ---
    ---
end <arq_name>;
```

Introduction to VHDL

Interface (entity):

- Define the access ports to the design.
- Basic generic form:

```
entity <entity_name> is
    port ( <signal1_name>: [mode] [type];
           <signal2_name>: [mode] [type] );
end < entity_name >;
```

- Possible modes of a port:

  in ⇒ the port is an input.

  out ⇒ the port is an output. The value of the associated signal cannot be used inside the architecture to define another signal. The position of the signal is always to the left of the assignment operator (<=).

  inout ⇒ the port may be used as input or output.

  buffer ⇒ the port is an output, but its value can be read inside the architecture. It can be to the left or to the right of the assignment operator (<=).

Introduction to VHDL

Implementation (architecture):

- Define the actual implementation of the design.
- Basic generic form:

```
architecture <arch_name> of <entity_name> is
    [signals declarations]
    [constants declarations]
    [types declarations]
    [components declarations]
    [attributes specifications]
begin
    [component instantiation]
    [concurrent assignment]
    [process]
    [generation]
end <arch_name>;
```

Introduction to VHDL

Example: 8 bits comparator

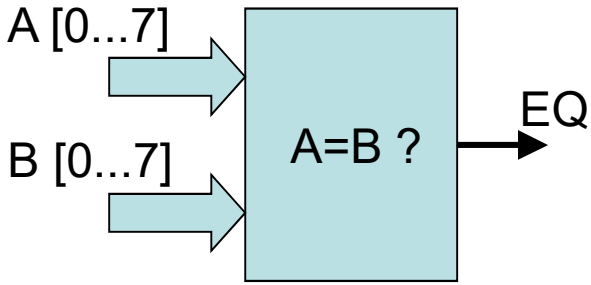
```
library ieee;

use ieee.std_logic_1164.all;

entity compare is
    port (A,B:   in std_logic_vector(0 to 7);
          EQ:   out std_logic);
end compare;

architecture one of compare is
begin
    EQ <= '1' when (A=B) else '0';
end one;
```

Schematics representation:



- The circuit is **combinational**.
- VHDL keywords in blue.
- VHDL is not *case-sensitive*.

Introduction to VHDL

VHDL keywords (*IEEE Std.1076-1993*)

abs	disconnect	label	package	sla
access	downto	library	port	sll
after	else	linkage	postponed	sra
alias	elsif	literal	procedure	srl
all	end	loop	process	subtype
and	entity	map	protected	then
architecture	exit	mod	pure	to
array	file	nand	range	transport
assert	for	new	record	type
attribute	function	next	register	unaffected
begin	generate	nor	reject	units
block	generic	not	rem	until
body	group	null	report	use
buffer	guarded	of	return	variable
bus	if	on	rol	wait
case	impure	open	ror	when
component	in	or	select	while
configuration	inertial	others	severity	with
constant	inout	out	shared	xnor
	is		signal	xor