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
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


- [www.altera.com](http://www.altera.com) (*datasheets, application notes, reference designs*)
- [www.xilinx.com](http://www.xilinx.com) (*datasheets, application notes, reference designs*)
- [www.doulos.com/knowhow/vhdl_designers_guide](http://www.doulos.com/knowhow/vhdl_designers_guide) (*The Designer’s Guide to VHDL*)
- [www.vhdl.org](http://www.vhdl.org)
Digital Electronics:

- logic gates
- flip-flops
- multiplexers
- comparators
- counters
- ...

Background required
Agenda

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

Vacuum tubes

Transistors

Integrated circuits
SSI, MSI, LSI, VLSI

1945
1950
1958
1990
today

SSI → Small-Scale Integration (<100 trans.)
MSI → Medium-Scale Integration
LSI → Large-Scale Integration (μp’s)
VLSI → Very Large-Scale Integration (>10<sup>6</sup> trans.)

ASIC → Application-Specific Integrated Circuit
PLD → Programmable Logic Device
DSP → Digital Signal Processor
μp → Microprocessor
μc → Microcontroller

Bipolar, CMOS, BiCMOS
### Digital Electronics: current technologies

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

(b) \(f_1 = x_1 \overline{x_2} + x_1 x_2\)

(c) Storage cell contents in the LUT
Example of logic block composed of a 3-inputs LUT and a register (D flip-flop).
Example of design

- LUT with 2 inputs.
- 4 wires of interconnection.
- 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.
Comparison of a low cost FPGA and a high performance FPGA:

<table>
<thead>
<tr>
<th></th>
<th>Cyclone IV family (1) (16 options)</th>
<th>Stratix IV family (1) (17 options)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>L=60 nm (1.2V or 1.0V core voltage)</td>
<td>L=40 nm (0.9V core voltage)</td>
</tr>
<tr>
<td>Logic Elements</td>
<td>6.272 - 149.760</td>
<td>72.600 - 813.050</td>
</tr>
<tr>
<td>Clock control</td>
<td>2 - 8 PLLs</td>
<td>3 - 12 PLLs</td>
</tr>
<tr>
<td>Memory blocks</td>
<td>270 – 6.480 kbits</td>
<td>6462 – 20.736 kbits</td>
</tr>
<tr>
<td>Multipliers</td>
<td>15 - 360 (18 bits X 18 bits)</td>
<td>384 - 1288 (18 bits X 18 bits)</td>
</tr>
<tr>
<td>Transceivers</td>
<td>2 - 8 (3,125 Gbps)</td>
<td>8 - 48 (11.3 Gbps)</td>
</tr>
<tr>
<td>I/O pins (max)</td>
<td>72 – 528</td>
<td>289 - 976</td>
</tr>
<tr>
<td>Programmable I/O</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Unit Price (US$)</td>
<td>from US$12 up to US$645</td>
<td>from US$800 up to US$24,270</td>
</tr>
</tbody>
</table>

(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
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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch-level modeling</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>ASIC timing</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Concurrency</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Design modularization</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Gate-level modeling</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Gate-level timing</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Four-state logic</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Event handling</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Basic data types</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Basic behavioral constructs</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Dynamic generation of hardware</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Configurations</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Simple assertions</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Assertions (formal methods)</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Dynamic-memory allocation</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Pointers</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Multidimensional arrays</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Records</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Enumeration</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Automatic variables</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Signed numbers</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>User-defined logic types</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User-defined resolution functions</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Void type</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Union</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Behavioral constructs</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Classes with methods and inheritance</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Sequential regular expressions</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Temporal-property definitions</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Scheduling for testbench and assertions</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Semaphores</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stimulus generation</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Constrained random data generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coverage monitoring</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Strings and strings operations</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Standard C interface</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Transaction modeling</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Module encryption</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
VHDL is a language to describe digital electronic circuits of medium to high complexity. Not recommended for simple designs.

The name stands 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 *functionality* of very complex integrated circuits.

Adopted and standardized by the *Institute of Electrical and Electronics Engineers* (IEEE).
• 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 functionality of performance of a design.
  – **Cost reduction** on prototyping.
  – **Time reduction** to insert a new product in the market.

• VHDL ⇒ Modeling - Simulation - **Synthesis**
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’ → *Initialized*  
  - ‘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.
Objects in VHDL

- There are 3 types of objects: **signals**, **constants** and **variables**.

- The name of an object can make use of any alphanumeric character, 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* characters 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:  
  ```vhdl
  signal <signal_name> : [type] ;
  ```

- The signal type defines its possible values and manipulation.
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: ```vhdlsignal 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 to use them, it must be included the following lines of code:
    ```vhdlslibrary ieee;
    use ieee.std_logic_1164.all;
    ```
  – values ‘0’, ‘1’, ‘Z’ and ‘X’ are the most useful for simulation and synthesis
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;
```
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;
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”;`
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)
FUNCTIONAL description:
• The electronic system is simply described by its function
  (ex: \( Y = (\overline{A} \text{ and } B) \) or \( (A \text{ and } \overline{B}) \)).
• 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:

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

architecture <arq_name> of <entity_name> is
begin
  ---
  ---
end <arq_name>;
```
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 (<=).
Implementation (architecture):

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

```vhdl
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>;
```
**Example: 8 bits comparator**

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