Introduction to VHDL

P. Albicocco

LNF-INFN

SUMMARY

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

VHDL → Very Hard Difficult Language ☺

VHDL → VHSIC Hardware Description Language

VHSIC → Very High Speed Integrated Circuits

- VHDL is used to model the physical hardware used in digital systems.
- VHDL allows most reliable design process minimizing both costs and develop time
- VHDL make use of Object Oriented methodology (modules developed for the current project can be reused in the future)

BUT

- VHDL is NOT a programming language (like C, C++, python etc.)

➡ when designing systems using VHDL you must always have in mind the hardware you have to implement

VHDL HISTORY

VHDL → VHSIC Hardware Description Language
VHSIC → Very High Speed Integrated Circuits

- VHDL is used for documentation, verification and synthesis of large digital design.
- VHDL allows hardware description using three different approaches: structural, data flow and behavioral (generally a mixture of the three methods is used).
- VHDL is a standard (VHDL-1076) developed by IEEE (Institute of Electrical and Electronics Engineers)

Summary: History of VHDL

1981	Initiated by US DoD to address hardware life-cycle crisis
1983-85	Development of baseline language by Intermetrics, IBM and TI
1986	All rights transferred to IEEE
1987	Publication of IEEE Standard
1987	Mil Std 454 requires comprehensive VHDL descriptions to be delivered with ASICs
1994	Revised standard (named VHDL 1076-1993)
2000	Revised standard (named VHDL 1076 2000, Edition)
2002	Revised standard (named VHDL 1076-2002)
2007	VHDL Procedural Language Application Interface standard (VHDL 1076c-2007)
2009	Revised Standard (named VHDL 1076-2008)

LEVELS OF ABSTRACTION

- Structural model:
  - The system is described as gates and component blocks connected by wires to implement the design (like a schematic rapresentation)
  - Describes only connections

- Behavioral model:
  - Describes the behavior of a component
  - Describes how input signal interact to create the output
  - Register Transfer Level (RTL):
    - Describes dataflow in the system
  - Algorithmic Level:
    - Instruction in a sequence of operations (sequential logic)
  - Dataflow model:
    - Define flow of data (example: x <= y is executed as soon as the input variable y change)
- VHDL respect the VLSI design principles of modularity and locality

SIMULATION AND SYNTHESIS

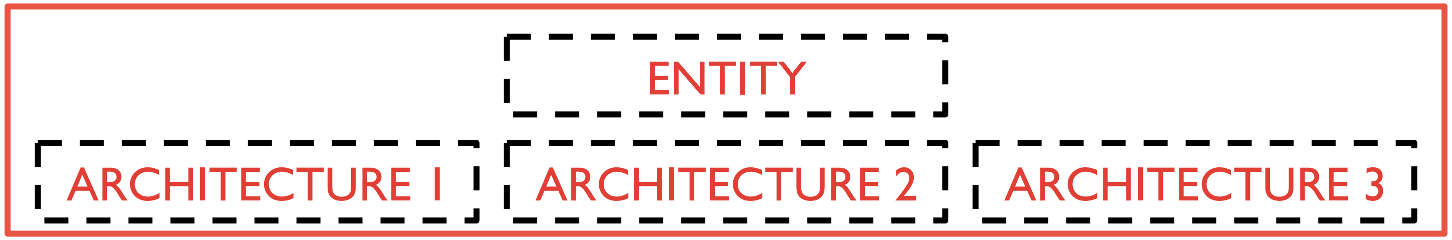
- Simulation is the prediction of the behavior of a system
  - Functional simulation generates an approximates behavior of the hardware design (functional simulation assume all outputs changing at the same time)
  - Timing simulation predicts the exact behavior of a hardware design
- Synthesis translates the design into a netlist file describing the hardware structure of a system
  - VHDL was not designed for synthesis
  - Not all VHDL statements are synthesizable

When starting a VHDL based design please remind Stephen Brown, Zvonko Vranesic [Fundamentals of Digital Logic with **VHDL** Design] suggestion:

"**A good general guideline is to assume that if the designer cannot readily determine what logic circuit is described by the VHDL code, then the CAD tools are not likely to synthesize the circuit that the designer is trying to describe**"

ENTITIES AND ARCHITECTURES

VHDL structure



ARCHITECTURES

- specify the design internal implementation
- An entity can have more architectures according to the required optimization (performance, area, power consumption, simulation)
- Configurations specify the connections between an architecture and an instance of an entity (in the following we'll use a single architecture)

ARCHITECTURES DECLARATIONS

```
ARCHITECTURE architecture_name OF entity_name IS
BEGIN
        -- Insert VHDL code here
        --
END architecture_name
```

ENTITIES

ENTITY = EXTERNAL INTERFACE SPECIFICATION

Entity declaration specify:

- The name of the entity
- A set of generic declarations defining the instance parameters
- A set of port declarations defining the inputs and outputs of the hardware design

EXAMPLE: OR GATE (F= x + y)

```
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
----------------------------------------
entity OR_ent is
port(x: in std_logic;
        y: in std_logic;
        F: out std_logic
);
end OR_ent;
----------------------------------------
```

Library: collection of design elements

entity – defines the interface

port direction; can be:
in, out, inout

Comment '-'

note: final signal has no semi-colon

ENTITY SECTION
→declare the I/O port of the circuit
define the names of the ports, their mode and their type

std_logic is the type of the port. Standard logic is defined by the standard IEEE 1164. It is defined in the IEEE library.
Any node of type std_logic can take 9 different values. '0' , '1' , 'H' , 'L' , 'Z' , 'U' , 'X' , 'W' , '-'

PORTS

PORT NAMES

- letters, digits, underscores
- begin with a letter
- are case insensitive

PORT DIRECTIONS

- in
- out
- inout
- buffer ← special OUT (can be read by the entity architecture)

IEEE standard 1164-1993

- the external pins of a synthesizable design must be defined using data types specified in the std_logic_1164 package
- IEEE strongly recommend the use of following data types for synthetizable system
  - std_logic
  - std_logic_vector(<max> DOWNTO <min>)

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

ARCHITECTURE

The ARCHITECTURE describes the behavior, interconnections and relationship between different inputs and outputs

CUNCURRENT SIGNAL ASSIGNMENT (CSA)

```
-----------------------------------------
architecture OR_beh of OR_ent is
begin
   F <= x or y;
end OR_beh;
-------------------------------------------
```

concurrent statement

PROCESS

```
-----------------------------------------
architecture OR_arch of OR_ent is
begin
   process(x, y)
   begin
      -- compare to truth table
      if ((x='0') and (y='0')) then
         F <= '0';
      else
         F <= '1';
      end if;
   end process;
end OR_arch;
-------------------------------------------
```

sensitivity list (process is invoked if the signal values change)

sequential statements

You can assign a name to the process

The use of processes makes your code more modular, more readable, and allows you to separate combinational logic from sequential logic

VARIABLES & SIGNALS

BOTH VARIABLES AND SIGNALS ARE USED TO HOLD DATA, BUT:

- Variables are used in processes (variables behave as expected in software programming languages)
- Signals are used in structural and data flow description

a:=b;

- variables do not trigger events
- variables are modified with the variable assignment ":="
- the value of b is copied immediately to a
- assignment is performed when the process is executed

VARIABLE EXAMPLE

```
count: process (x)
  variable cnt : integer := -1;
begin
  cnt:=cnt+1;
end process;
```

- variable declaration appears BEFORE the begin keyword (optional: the initial value can be specified and will be used in simulation)
- Variable cnt is declared to be of the type integer (can assume both positive and negative values)

each time the process is executed the value cnt+1 is stored in cnt variable (as process is executed once when simulation start initializing cnt to -1 allows the value of cnt be 0 when simulation start)

cnt is incremented each time the signal (x) in the sensitivity list change, then if x is a bit signal the process counts the number of rising and falling edges of the signal

SYNTHESIS AND SIMULATION: BEWARE

- Synthesizable design can be simulated
- Not all simulated signals can be synthetized

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY simple_buffer IS
    PORT ( din : IN std_logic;
           dout : OUT std_logic
    );
END simple_buffer;
ARCHITECTURE behavioural1 OF simple_buffer IS
BEGIN
    dout <= din AFTER 10 ns;
END behavioural1;
```

THIS ARCHITECTURE CAN BE SIMULATED BUT NOT SYNTETIZED

VHDL OPERATORS

LOGICAL OPERATORS

"AND"
"OR"
"NAND"
"NOR"
"XOR"

RELATIONAL OPERATORS

=	(EQUAL)
/=	(NOT EQUAL)
<	(LESS THAN)
>	(GREATER THAN)

MATHEMATICAL OPERATORS

+	(ADDITION)
-	(SUBTRACTION)
*	(MULTIPLICATION)
/	(DIVISION)

ASSIGNMENT STATEMENTS EXAMPLE

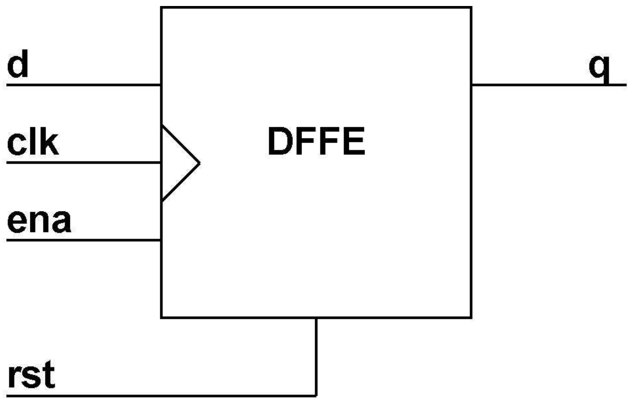
```
SIGNAL a, b, c : std_logic;
SIGNAL avec, bvec, cvec : std_logic_vector(7 DOWNTO 0);

-- Concurrent Signal Assignment Statements
-- NOTE: Both a and avec are produced concurrently
a       <= b AND c;
avec    <= bvec OR cvec;

-- Alternatively, signals may be assigned constants
a       <= ’0’;
b       <= ’1’;
c       <= ’Z’;
avec    <= "00111010";      -- Assigns 0x3A to avec
Bvec     <= X"3A”;          -- Assigns 0x3A to bvec
cvec    <= X"3" & X"A";     -- Assigns 0x3A to cvec
```

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

PROCESS: D FF EXAMPLE

D FLIP-FLOP BLOCK DIAGRAM

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY dff IS
        PORT(rst, clk, ena, d : IN      std_logic
                   q :           OUT     std_logic
END dff;
ARCHITECTURE synthesis1 OF dff IS
BEGIN
        PROCESS (rst, clk)
        BEGIN
                IF (rst = ‘1’) THEN
                        q <= ‘0’;
                ELSIF (clk’EVENT) AND (clk = ‘1’) THEN
                        IF (ena = ‘1’) THEN
                                q <= d;
                        END IF;
                END IF;
        END PROCESS;
END synthesis1;
```



EVENT attribute is used to check the rising edge of a clock signal

CONCURRENT STATEMENTS

- All concurrent statements in an architecture are executed simultaneously
- Concurrent statements are used to express parallel activity
- Concurrent statements are executed with no predefined order by the simulator . So the order in which the code is written does not have any effect on its function
- Process is a concurrent statement in which sequential statements are allowed
- All processes in an architecture are executed simultaneously
- Concurrent statements are executed by the simulator when one of the signals in its sensitivity list changes .This is called occurrence of 'event' (c <= a or b; is executed when either signal 'a' or signal 'b' changes

VHDL INSTRUCTIONS

VHDL_INSTRUCTION FILE CONTAINS DESCRIPTION AND EXAMPLES OF FPGA BUILDING BLOCKS AND INSTRUCTIONS

LET'S START WRITING SOME CODE …

REMIND

- Every VHDL design description consists of at least one *entity* / *architecture* pair, or one entity with multiple architectures.
- The entity section of the HDL design is used to declare the *I/O ports* of the circuit, while the description code resides within architecture portion.
- Standardized design libraries are typically used and are included prior to the entity declaration. This is accomplished by including the code "library ieee;" and "use ieee.std_logic_1164.all;".

AND GATE

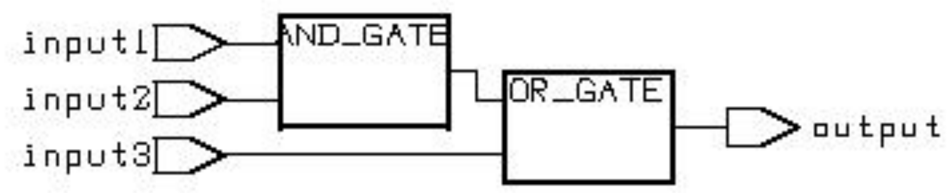
```
--------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------
entity AND_ent is
port(   x: in std_logic;
        y: in std_logic;
        F: out std_logic
);
end AND_ent;
--------------------------------------------------
architecture behav1 of AND_ent is
begin
   process(x, y)
   begin
      -- compare to truth table
      if ((x='1') and (y='1')) then
           F <= '1';
        else
           F <= '0';
        end if;
   end process;
end behav1;
```

```
--------------------------------------------------
architecture behav2 of AND_ent is
begin
   F <= x and y;
end behav2;
```

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

COMBINATIONAL LOGIC

The port map instruction is used for component instantiation. The program incorporates multiple components in the design.



OR

```
-------------------------------------------------------------

library ieee;                           -- component #1
use ieee.std_logic_1164.all;
entity OR_GATE is
port(   X:      in std_logic;
        Y:      in std_logic;
        F2:     out std_logic
);
end OR_GATE;
architecture behv of OR_GATE is
begin
process(X,Y)
begin
        F2 <= X or Y;                   -- behavior des.
end process;
end behv;
```

AND

```
-------------------------------------------------------------

library ieee;                           -- component #2
use ieee.std_logic_1164.all;
entity AND_GATE is
port(   A:      in std_logic;
        B:      in std_logic;
        F1:     out std_logic
);
end AND_GATE;
architecture behv of AND_GATE is
begin
process(A,B)
begin
        F1 <= A and B;                  -- behavior des.
end process;
end behv;
```

COMBINATIONAL LOGIC

```
------------------------------------------------------------
library ieee;                           -- top level circuit
use ieee.std_logic_1164.all;
use work.all;
entity comb_ckt is
port(   input1: in std_logic;
        input2: in std_logic;
        input3: in std_logic;
        output: out std_logic
);
end comb_ckt;
architecture struct of comb_ckt is
   component AND_GATE is                -- as entity of AND_GATE
   port(   A:   in std_logic;
           B:   in std_logic;
          F1:   out std_logic
   );
   end component;
   component OR_GATE is                 -- as entity of OR_GATE
   port(   X:   in std_logic;
           Y:   in std_logic;
           F2: out std_logic
   );
   end component;
   signal wire: std_logic;              -- signal just like wire
begin
   -- use sign "=>" to clarify the pin mapping
   Gate1: AND_GATE port map (A=>input1, B=>input2, F1=>wire);
   Gate2: OR_GATE port map (X=>wire, Y=>input3, F2=>output);
end struct;
------------------------------------------------------------
```

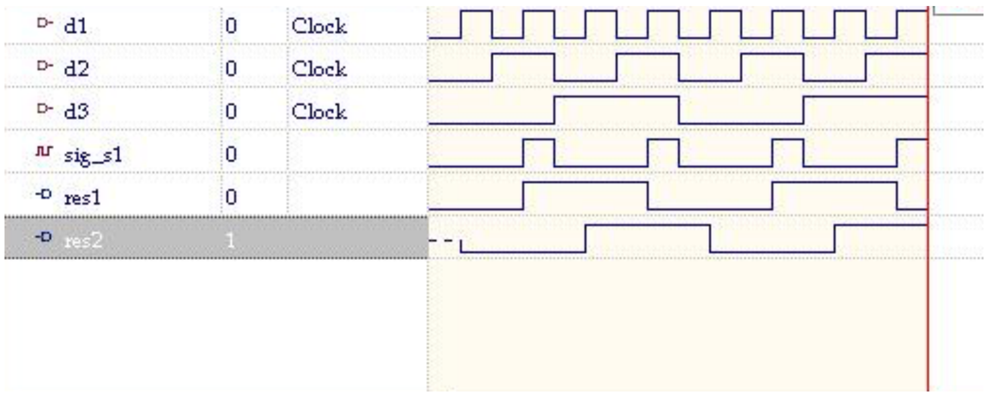
COMPONENTS INSTANTIATION

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

SIGNALS AND VARIABLES

Signals are used to connect the design components and must carry the information between current statements of the design. On the other hand, variables are used within process to compute certain values.

```
library ieee;
use ieee.std_logic_1164.all;
entity sig_var is
port(   d1, d2, d3:     in std_logic;
        res1, res2:     out std_logic);
end sig_var;
architecture behv of sig_var is
  signal sig_s1: std_logic;
Begin
  proc1: process(d1,d2,d3)
    variable var_s1: std_logic;
  begin
        var_s1 := d1 and d2;
        res1 <= var_s1 xor d3;
  end process;
  proc2: process(d1,d2,d3)
  begin
        sig_s1 <= d1 and d2;
        res2 <= sig_s1 xor d3;
  end process;
end behv;
```



The first process uses variables while the second uses signals: the outputs of the two processes are different !!!

COMBINATIONAL COMPONENTS

- The body of an architecture includes concurrent signal assignment, concurrent processes and component instantiation (port map statements)
- Process statements include sequential statements (case/if-then-else/loop)

```
library ieee;
use ieee.std_logic_1164.all;
--------------------------------------------------
entity DECODER is
port(   I:      in std_logic_vector(1 downto 0);
        O:      out std_logic_vector(3 downto 0)
);
end DECODER;
--------------------------------------------------
architecture behv of DECODER is
begin
   -- process statement
   process (I)
   begin
      -- use case statement
      case I is
            when "00" => O <= "0001";
            when "01" => O <= "0010";
            when "10" => O <= "0100";
            when "11" => O <= "1000";
            when others => O <= "XXXX";
        end case;
   end process;
end behv;
```

Concurrent statement (with sensitivity list)

Sequential statements inside a process (you could use if-then-else as well but case statement looks micer)

When using case statement you have to specify what happen in 'other cases'

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

SEQUENTIAL COMPONENTS

Flip-flop is a basic component of sequential circuits. Besides input/output signals flip-flop requires the reset and the clock signals (reset can be active high or active low flip-flop transitions can occur both at the clock *rising-edge* or *falling edge*)

```
-----------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
-----------------------------------------------
entity JK_FF is
port (  clock:        in std_logic;
        J, K:         in std_logic;
        reset:        in std_logic;
        Q, Qbar:      out std_logic
);
end JK_FF;
-----------------------------------------------
```

```
-----------------------------------------------
architecture behv of JK_FF is
   -- define the useful signals here
   signal state: std_logic;
   signal input: std_logic_vector(1 downto 0);
begin
   -- combine inputs into vector
   input <= J & K;
   p: process(clock, reset) is
   begin
        if (reset='1') then
           state <= '0';
        elsif (rising_edge(clock)) then
         -- compare to the truth table
           case (input) is
                when "11" =>
                   state <= not state;
                when "10" =>
                   state <= '1';
                when "01" =>
                   state <= '0';
                when others =>
                   null;
                end case;
        end if;
   end process;
```

```
-- concurrent statements
   Q <= state;
   Qbar <= not state;
end behv;
-----------------------------------------------
```

reset active high

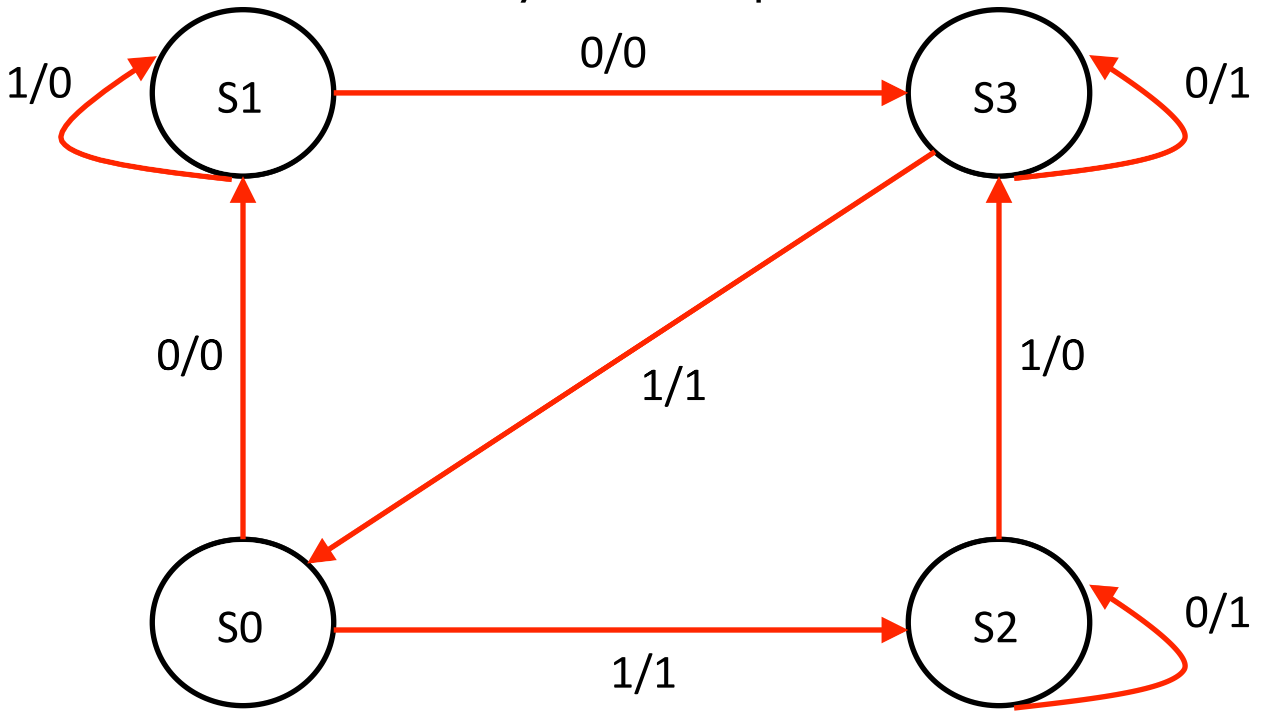
Clock rising edge

sequential statement inside a process

SEQUENTIAL DESIGN (I)

Finite State Machine (FSM) is a key component of VHDL design. It consist of both combinational logic and sequential components. Sequential components (registers) are used to record the state of the circuit and are updated synchronously on the rising edge of the clock signal.
Two types of state machine: Moore (outputs depend on current state only) and Mealy (outputs depend on current state and inputs).

Mealy FSM example



EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

SEQUENTIAL DESIGN (2)

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

entity mealy is
port (clk : in std_logic;
      reset : in std_logic;
      input : in std_logic;
      output : out std_logic
  );
end mealy;

architecture behavioral of mealy is

type state_type is (s0,s1,s2,s3);

signal current_s,next_s: state_type;
```

```
begin

process (clk,reset)
begin
 if (reset='1') then
  current_s <= s0;
elsif (rising_edge(clk)) then
  current_s <= next_s;
end if;
end process;
```

type of state machine

current and next-state declaration

default state on reset

state change

EDIT 2015
INTERNATIONAL SCHOOL
FRASCATI - OCT.20-29

SEQUENTIAL DESIGN (3)

```
process (current_s,input)
begin
 case current_s is
   when s0 =>
   if(input ='0') then
    output <= '0';
    next_s <= s1;
  else
    output <= '1';
    next_s <= s2;
   end if;
   when s1 =>;
  if(input ='0') then
    output <= '0';
    next_s <= s3;
  else
    output <= '0';
    next_s <= s1;
  end if;
```

when current state is s0

when current state is s1

```
  when s2 =>
   if(input ='0') then
     output <= '1';
     next_s <= s2;
   else
     output <= '0';
     next_s <= s3;
   end if;
 when s3 =>
  if(input ='0') then
     output <= '1';
     next_s <= s3;
   else
     output <= '1';
     next_s <= s0;
   end if;
 end case;
end process;
end behavioral
```

when current state is s2

when current state is s3

CONCLUSIONS

- This talk has only covered a small part of the VHDL language; many constructs and features of state of the art FPGA have been omitted (memory, high speed serial links, processor implementation ….)
- For those interested in learning the VHDL language a good book together with a Xilinx or Altera supported evaluation board could be the next step
- If the choice is for the XLINX products a good starting point could be an evaluation board based on Zynq FPGA e.g. the ZedBoard, a development kit based on Xilinx Zynq-7000 All Programmable SoC.

REMIND

WHEN WRITING VHDL CODE YOU HAVE ALWAYS KEEP IN MIND THAT YOU ARE DESIGNING HARDWARE, NOT A COMPUTER PROGRAM !!!

EDIT 2015 INTERNATIONAL SCHOOL FRASCATI - OCT.20-29

BIBLIOGRAPHY

[1] Introduction to VHDL - Michael Lupberger - University of Bonn
[2] VHDL tutorial - William D. Bishop - Department of Electrical and Computer Engineering University of Waterloo
[3] http://esd.cs.ucr.edu/labs/tutorial/