Introduction to VHDL
Based on Altera’s Tutorial

Computer Architecture

Prof. Erivelton G. Nepomuceno
nepomuceno@ufsj.edu.br
Course Outline

- VHDL Basics
- Design Units
- Architecture Modeling Fundamentals
- Understanding VHDL and Logic Synthesis
- Hierarchical Designing
VHDL Basics

- IEEE industry standard hardware description language
- High-level description language for both Simulation & Synthesis
- 1980 - U.S. Department of Defense (DOD) funded a project to create a standard hardware description language under the Very High Speed Integrated Circuit (VHSIC) program.
- 1987 - the Institute of Electrical and Electronics Engineers (IEEE) ratified as IEEE Standard 1076.
- 1993 - the VHDL language was revised and updated to IEEE 1076 ‘93.
Terminology

- **HDL - Hardware Description Language** is a software programming language that is used to model a piece of hardware.

- **Behavior Modeling** - A component is described by its input/output response.

- **Structural Modeling** - A component is described by interconnecting lower-level components/primitives.
Terminology

- Register Transfer Level (RTL) - A type of behavioral modeling, for the purpose of synthesis.
  - Hardware is implied or inferred
  - Synthesizable

- Synthesis - Translating HDL to a circuit and then optimizing the represented circuit

- Process – Basic unit of execution in VHDL
Behavior Modeling

- Only the functionality of the circuit, no structure
- No specific hardware intent
- For the purpose of synthesis, as well as simulation

```
IF input1 THEN
    FOR j IN high DOWNTO low LOOP
        shft(j) := shft(j);
    END LOOP;
output1 <= shft AFTER 5ns;
```
Structural Modeling

- Functionality and structure of the circuit
- Call out the specific hardware
- For the purpose of synthesis
Process (a, b, c, d, sel)
begin
  case (sel) is
    when "00" => mux_out <= a;
    when "01" => mux_out <= b;
    when "10" => mux_out <= c;
    when "11" => mux_out <= d;
  end case;
end;
VHDL Synthesis vs. Other HDL Standards

- **VHDL**
  - “Give me a circuit whose output only changes when there is a low-to-high transition on a particular input. When the transition happens, make the output equal to the input until the next transition.”
  - Result: VHDL Synthesis provides a positive edge-triggered flipflop

- **ABEL, PALASM, AHDL**
  - “Give me a D-type flipflop.”
  - Result: ABEL, PALASM, AHDL synthesis provides a D-type flipflop. The sense of the clock depends on the synthesis tool.
More VHDL Basics

- Two sets of constructs:
  - Synthesis
  - Simulation

- The VHDL Language is made up of reserved keywords.

- The language is, for the most part, NOT case sensitive.

- VHDL statements are terminated with a ;

- VHDL is white space insensitive. Used for readability.

- Comments in VHDL begin with “--” to eol
Typical Synthesis Design Flow

- VHDL Library
- VHDL Model
- Technology Library
- Synthesis Compiler
- Netlist
- Timing Analysis
- Place/Route
- Text Output
- Waveform
- Simulation
- Test Vectors
Typical Simulation Design Flow

- VHDL Library
- VHDL Model
- VHDL TestBench

Simulation Compiler

- Simulation Model
- Test Vectors

VHDL Simulation

Waveform

Text Output
VHDL
Design Units
Design Units

- **VHDL Design Units**
  - **Entity**
    - Used to define external view of a model. I.e. symbol
  - **Architecture**
    - Used to define the function of the model. I.e. schematic
  - **Configuration**
    - Used to associate an Architecture with an Entity
  - **Package**
    - Collection of information that can be referenced by VHDL models. I.e. Library
    - Consist of two parts Package Declaration and Package Body.
Entity Declaration

**ENTITY** `<entity_name>` **IS**
- Generic Declarations
- Port Declarations

**END** `<entity_name>`; (1076-1987 version)

**END ENTITY** `<entity_name>` ; (1076-1993 version)

- **Analogy** : Symbol
- `<entity_name>` can be any alpha/numerical name
  - Note: MAX+PLUS II requires that the `<entity_name>` and `<file_name>` be the same
- **Generic Declarations**
  - Used to pass information into a model
  - MAX+PLUS II places some restriction on the use of Generics
- **Port Declarations**
  - Used to describe the inputs and outputs i.e. pins
Entity: Generic Declaration

ENTITY <entity_name> IS

Generic (constant tplh, tphl : time := 5 ns

-- Note constant is assumed and is not required
tphz, tplz : time := 3 ns;
default_value : integer := 1;
cnt_dir : string := “up”
);

Port Declarations

END <entity_name>; (1076-1987 version)
END ENTITY <entity_name> ; (1076-1993 version)

- New values can be passed during compilation
- During simulation/synthesis a Generic is read only
**Entity : Port Declarations**

**ENTITY** `<entity_name>` **IS**

Generic Declarations

Port (signal clk : **in** bit;
    --Note: **signal** is assumed and is not required
    q **out** bit)

);

END `<entity_name>`; (1076-1987 version)

END ENTITY `<entity_name>` ; (1076-1993 version)

- **Structure**: `<class>` object_name : `<mode>` `<type>` ;
  - `<class>` : what can be done to an object
  - Object_name : identifier
  - `<mode>` : directional
    - **in** (input) **out** (output)
    - **inout** (bidirectional) **buffer** (output w/ internal feedback)
  - `<type>` : What can be contained in the object
Architecture

Key aspects of the Architecture

- Analogy: schematic
- Describes the Functionality and Timing of a model
- Must be associated with an ENTITY
- ENTITY can have multiple architectures
- Architecture statements execute concurrently
- Architecture Styles
  - Behavioral: How designs operate
    - RTL: Designs are described in terms of Registers
    - Functional: No timing
  - Structural: Netlist
    - Gate/Component Level
  - Hybrid: Mixture of the above
Configuration

- Used to make associations within models
  - Associate a Entity and Architecture
  - Associate a component to an Entity-Architecture
- Widely used in Simulation environments
  - Provides a flexible and fast path to design alternatives
- Limited or no support in Synthesis environments

```
CONFIGURATION <identifier> OF <entity_name> IS
  FOR <architecture_name>
  END FOR;
END; (1076-1987 version)
END CONFIGURATION; (1076-1993 version)
```
Testbench

Testbench is not defined by the VHDL Language Reference Manual and has no formal definition.

In general, it consists of three parts:

1. The component we want to test, i.e. the Design Under Test (DUT).
2. A mechanism for supplying inputs to the DUT.
3. A mechanism for checking the outputs of the DUT against expected outputs.
Testbench architecture, Source: Kashani-Akhavan. Available at https://goo.gl/dCsMNK
ENTITY cmpl_sig IS
PORT (a, b, sel : IN bit;
     x, y, z : OUT bit;
END cmpl_sig;
ARCHITECTURE logic OF cmpl_sig IS
BEGIN
    -- simple signal assignment
    x <= (a AND NOT sel) OR (b AND sel);
    -- conditional signal assignment
    y <= a WHEN sel='0' ELSE b;
    -- selected signal assignment
    WITH sel SELECT
        z <= a WHEN '0',
            b WHEN '1',
            '0' WHEN OTHERS;
END logic;
CONFIGURATION cmpl_sig_conf OF cmpl_sig IS
    FOR logic
    END FOR;
END cmpl_sig_conf;
Example 1

-- Exemplo 1 - Porta AND - 2 entradas
-- Arquitetura de Computadores
-- Prof. Erivelton

library ieee;
use ieee.std_logic_1164.all;

entity PortaAND2to1 is
  port
  (    a : in std_logic;
      b : in std_logic;
      saída : out std_logic
  );
end entity;

architecture rtl of PortaAND2to1 is
begin
  saída <= a and b;
end rtl;
Example 1

- Compilation using Quartus II

![Quartus II screenshot](image)
Example 1

- Compilation using Quartus II

![Image of Quartus II software interface](image_url)
Example 1

**Directory, Name, Top-Level Entity [page 1 of 5]**

What is the **working** directory for this project?

C:/Users/user/Google Drive/teaching/201801/arqcomp/vhdl/quartus

What is the name of this project?

PortaAND2to1

What is the name of the top-level design entity for this project? This name is case sensitive.

PortaAND2to1

*Use Existing Project Settings...*
Example 1

Add Files [page 2 of 5]

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.
Note: you can always add design files to the project later.

File name:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Type</th>
<th>Library</th>
<th>Design Entry/Synthesis Tool</th>
<th>HDL Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>..modelsim/Exemplo1_PortaAND2to1/PortaAND2to1.vhd</td>
<td>VHD...</td>
<td></td>
<td></td>
<td>Default</td>
</tr>
</tbody>
</table>

Add All
Add
Remove
Up
Example 1

New Project Wizard

Family & Device Settings [page 3 of 5]

Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

Device family

- Family: Cyclone IV E
- Device: Cyclone II
- Target device: Cyclone IV E

Show in 'Available devices' list

- Package: Any
- Pin count: Any
- Speed grade: Any

Available devices:

<table>
<thead>
<tr>
<th>Name</th>
<th>Core Voltage</th>
<th>LEs</th>
<th>User I/Os</th>
<th>Memory Bits</th>
<th>Embedded multiplier 9-bit elements</th>
<th>PLL</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP4CE6E22A7</td>
<td>1.2V</td>
<td>6272</td>
<td>92</td>
<td>275-480</td>
<td>30</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>EP4CE6E22C6</td>
<td>1.2V</td>
<td>6272</td>
<td>92</td>
<td>275-480</td>
<td>30</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>EP4CE6E22C7</td>
<td>1.2V</td>
<td>6272</td>
<td>92</td>
<td>275-480</td>
<td>30</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>EP4CE6E22C8</td>
<td>1.2V</td>
<td>6272</td>
<td>92</td>
<td>275-480</td>
<td>30</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>EP4CE6E22CL</td>
<td>1.0V</td>
<td>6272</td>
<td>92</td>
<td>275-480</td>
<td>30</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Companion device

- HardCopy:
- Limit DSP & RAM to HardCopy device resources

Copyright © 2003 Altera Corporation
Example 1

EDA Tool Settings [page 4 of 5]

Specify the other EDA tools used with the Quartus II software to develop your project.

EDA tools:

<table>
<thead>
<tr>
<th>Tool Type</th>
<th>Tool Name</th>
<th>Format(s)</th>
<th>Run Tool Automatically</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Entry/Synthesis</td>
<td>&lt;None&gt;</td>
<td>&lt;None&gt;</td>
<td>Run this tool automatically to synthesize the current design</td>
</tr>
<tr>
<td>Simulation</td>
<td>ModelSim-Altera</td>
<td>VHDL</td>
<td>Run gate-level simulation automatically after compilation</td>
</tr>
<tr>
<td>Formal Verification</td>
<td>&lt;None&gt;</td>
<td>&lt;None&gt;</td>
<td></td>
</tr>
<tr>
<td>Board-Level</td>
<td>Timing</td>
<td>&lt;None&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symbol</td>
<td>&lt;None&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Signal Integrity</td>
<td>&lt;None&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Boundary Scan</td>
<td>&lt;None&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Example 1

Summary [page 5 of 5]

When you click Finish, the project will be created with the following settings:

- Project directory: C:/Users/user/Google Drive/teaching/20J801/arqcomp/vhdl/quartus
- Project name: PortsAND2to1
- Top-level design entity: PortsAND2to1
- Number of files added: 1
- Number of user libraries added: 0
- Device assignments:
  - Family name: Cyclone IV E
  - Device: AUTO
- EDA tools:
  - Design entry/synthesis: <None> (<None>)
  - Simulation: ModelSim-Altera (VHDL)
  - Timing analysis: 0
- Operating conditions:
  - Core voltage: n/a
  - Junction temperature range: n/a
Example 1
Example 1
Example 1

Diagram of a circuit with inputs a and b connected to an AND gate labeled "saida" with output "saida".
Example 1 - Testbench

-- Exemplo 1 - Porta AND - 2 entradas – Testbench
-- Arquitetura de Computadores
-- Prof. Erivelton

library ieee;
use ieee.std_logic_1164.all;

entity tb_PortaAND2to1 is
end tb_PortaAND2to1;

architecture behavior of tb_PortaAND2to1 is
-- Declaração de componente de: Unit Under Test (UUT)

component PortaAND2to1
    port
    (a : in std_logic;
     b : in std_logic;
     a saída : out std_logic
    );
Example 1 – Testbench - Continued

-- Sinais
signal a : std_logic;
signal b : std_logic;
signal saida : std_logic;

begin
-- Definição the Unit Under Test (UUT)
uut: PortaAND2to1 port map
(    a => a,
    b => b,
    saida => saida
);

Example 1 – Testbench - Continued

-- Stimulus process
stim_proc: process
begin
   -- insert stimulus here
   wait for 5 ns;
a <= '0';
b <= '0';
wait for 5 ns;
a <= '1';
b <= '0';
wait for 5 ns;
a <= '0';
b <= '1';
wait for 5 ns;
a <= '1';
b <= '1';
wait;
end process;
end architecture;
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim

- Add Existing File
- Create New File
- Create Simulation
- Create New Folder
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim
Example 1 - Modelsim

Simulation Time

Simulation

Full View
Example 1 - ModelSim
Example 1 – Modelsim - Script

```vhd
## NOTE: Do not edit this file.
#
if {[file exists work]} {
  vdel -lib work -all
}
vlib work
vcom -explicit -93 "PortaAND2to1.vhd"
vcom -explicit -93 "tb_PortaAND2to1.vhd"
vsim -t 1ps -lib work tb_PortaAND2to1
#add wave sim:/tb_PortaAND2to1/*
do {wave.do}
view wave
view structure
view signals
run 0.03us
#quit -force
```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /tb_portaand2to1/a
add wave -noupdate /tb_portaand2to1/b
add wave -noupdate /tb_portaand2to1/saida
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {0 ps} 0}
quietly wave cursor active 1
configure wave -namecolwidth 81
configure wave -valuecolwidth 45
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ns
update
WaveRestoreZoom {0 ps} {31500 ps}
Example 1 – Modelsim - Script
Example 1 – Modelsim - Script
Example 1 – Modelsim - Script

```
do tb_my_project.do
```

```
ModelSim> do tb_my_project.do
```

```
0 ps to 36 ns
```
Example 1 – Modelsim - Script
Packages

- Packages are a convenient way of storing and using information throughout an entire model.

- Packages consist of:
  - Package Declaration (Required)
    - Type declarations
    - Subprograms declarations
  - Package Body (Optional)
    - Subprogram definitions

- VHDL has two built-in Packages
  - Standard
  - TEXTIO
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE filt_cmp IS
  TYPE state_type IS (idle, tap1, tap2, tap3, tap4);
  COMPONENT acc
    port(xh : in std_logic_vector(10 downto 0);
         clk, first: in std_logic;
         yn : out std_logic_vector(11 downto 4));
  END COMPONENT;
  FUNCTION compare (variable a, b : integer) RETURN boolean;
END filt_cmp;

PACKAGE BODY filt_cmp IS
  FUNCTION compare (variable a, b : integer) IS
    VARIABLE temp : boolean;
    Begin
      If a < b then
        temp := true;
      else
        temp := false;
      end if;
      RETURN temp;
    END compare;
END filt_cmp;
Libraries

- Contains a package or a collection of packages
- Resource Libraries
  - Standard Package
  - IEEE developed packages
  - Altera Component packages
  - Any library of design units that are referenced in a design
- Working Library
  - Library into which the unit is being compiled
All packages must be compiled

Implicit Libraries
- Work
- STD

⇒ Note: Items in these packages do not need to be referenced, they are implied

• LIBRARY Clause
  - Defines the library name that can be referenced
  - Is a symbolic name to path/directory
  - Defined by the Compiler Tool

• USE Clause
  - Specifies the package and object in the library that you have specified in the Library clause
LIBRARY iee;
USE iee.std_logic_1164.all;
ENTITY cmpl_sig IS
PORT ( a, b, sel : IN std_logic;
       x, y, z : OUT std_logic;
END cmpl_sig;
ARCHITECTURE logic OF cmpl_sig IS
BEGIN
  -- simple signal assignment
  x <= (a AND NOT sel) OR (b AND sel);
  -- conditional signal assignment
  y <= a WHEN sel='0' ELSE b;
  -- selected signal assignment
  WITH sel SELECT
    z <= a WHEN '0',
        b WHEN '1',
        '0' WHEN OTHERS;
END logic;
CONFIGURATION cmpl_sig_conf OF cmpl_sig IS
  FOR logic
END FOR;
END cmpl_sig_conf;

- **LIBRARY** <name>, <name> ;
  - name is symbolic and define by compiler tool
  - Note: Remember that WORK and STD do not need to be defined.

- **USE** lib_name.pack_name.object;
  - **ALL** is a reserved word

- Placing the Library/Use clause 1st will allow all following design units to access it
Libraries

```vhdl
LIBRARY STD;

-- Contains the following packages:
  • standard (Types: Bit, Boolean, Integer, Real, and Time. All operator functions to support types)
  • textio (File operations)

-- An implicit library (built-in)
  • Does not need to be referenced in VHDL design
```
Types Defined in Standard Package

- **Type BIT**
  - 2 logic value system (‘0’, ‘1’)
    ```
    signal a_temp : bit;
    ```
  - BIT_VECTOR array of bits
    ```
    signal temp : bit_vector(3 downto 0);
    signal temp : bit_vector(0 to 3);
    ```

- **Type BOOLEAN**
  - (false, true)

- **Integer**
  - Positive and negative values in decimal
    ```
    signal int_tmp : integer; -- 32 bit number
    signal int_tmp1 : integer range 0 to 255; -- 8 bit number
    ```

⇒ Note: Standard package has other types
Libraries

- LIBRARY IEEE;
  - Contains the following packages:
    - std_logic_1164 (std_logic types & related functions)
    - std_logic_arith (arithmetic functions)
    - std_logic_signed (signed arithmetic functions)
    - std_logic_unsigned (unsigned arithmetic functions)
Types Defined in std_logic_1164 Package

- **Type STD_LOGIC**
    - ‘W’, ‘L’, ‘H’ weak values (Not supported by Synthesis)
    - ‘X’ - used for unknown
    - ‘Z’ - (not ‘z’) used for tri-state
    - ‘-’ Don’t Care
  - Resolved type: supports signals with multiple drives

- **Type STD_ULOGIC**
  - Same 9 value system as STD_LOGIC
  - Unresolved type: Does not support multiple signal drives; Error will occur
## VHDL Operators

<table>
<thead>
<tr>
<th>Operator Type</th>
<th>Operator Name/Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>and or nand nor xor xnor(1)</td>
</tr>
<tr>
<td>Relational</td>
<td>= /= &lt; &lt;= &gt; &gt;=</td>
</tr>
<tr>
<td>Adding</td>
<td>+ - &amp;</td>
</tr>
<tr>
<td>Signing</td>
<td>+ -</td>
</tr>
<tr>
<td>Multiplying</td>
<td>* / mod rem</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>** abs not</td>
</tr>
</tbody>
</table>

(1) Supported in VHDL '93 only
Operator Overloading

How do you use Arithmetic & Boolean functions with other data types?

- **Operator Overloading** - defining Arithmetic & Boolean functions with other data types

Operators are overloaded by defining a function whose name is the same as the operator itself

- Because the operator and function name are the same, the function name must be enclosed within double quotes to distinguish it from the actual VHDL operator
- The function is normally declared in a package so that it is globally visible for any design
Review

- Terminology
  - Synthesis
  - Behavior Modeling
  - Structural Modeling

- Design Units
  - Entity
  - Architecture
  - Configuration
  - Package

- Libraries
  - work
  - ieee
Architecture Modeling Fundamentals
Using Signals

- Signals represent physical interconnect (wire) that communicate between processes (functions)
- Signals can be declared in **Packages**, **Entity** and **Architecture**
Assigning Values to Signals

SIGNAL temp : STD_LOGIC_VECTOR (7 downto 0);

- All bits:
  temp <= "10101010";
  temp <= x"AA" ; (1076-1993)

- Single bit:
  temp(7) <= '1';

- Bit-slicing:
  temp (7 downto 4) <= "1010";

- Single-bit: single-quote ('')
- Multi-bit: double-quote (""
Signal Used As an Interconnect

LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY simp IS
PORT(r, t, g, h : IN STD_LOGIC;
        qb : OUT STD_LOGIC);
END simp;
ARCHITECTURE logic OF simp IS
SIGNAL qa : STD_LOGIC;
BEGIN
  qa <= r or t;
  qb <= (qa and not(g xor h));
END logic;

r, t, g, h, and qb are Signals (by default)
qa is a buried Signal and needs to be declared

Signal Declaration inside Architecture
Simple Signal Assignments

Format: `<signal_name> <= <expression>;`

Example:

```
qa <= r or t;
qb <= (qa and not(g xor h));
```

VHDL Operators are used to describe the process.

Parentheses ( ) give the order of operation.
Conditional Signal Assignments

Format:

```
<signal_name>  <=  <signal/value>  when  <condition1>  else
<signal/value>  when  <condition2>  else
  ...
<signal/value>  when  <condition3>  else
<signal/value>;
```

Example:

```
q  <=  a  WHEN  sela  =  '1'  ELSE
    b  WHEN  selv  =  '1'  ELSE
    c;
```

Implied process: 

```
sela
 selb
   a
     c
   b
     q
```

Copyright © 2003 Altera Corporation
Selected Signal Assignments

Format:
with <expression> select
<signal_name> <= <signal/value> when <condition1>,
<signal/value> when <condition2>,
.
.
<signal/value> when others;

Example:
WITH sel SELECT
q <= a WHEN "00",
b WHEN "01",
c WHEN "10",
d WHEN OTHERS;

implied process
If-Then Statements

Format:

IF <condition1> THEN
   {sequence of statement(s)}
ELSIF <condition2> THEN
   {sequence of statement(s)}
   .
   .
ELSE
   {sequence of statement(s)}
END IF;

Example:

PROCESS(sela, selb, a, b, c)
BEGIN
   IF sela='1' THEN
      q <= a;
   ELSIF selb='1' THEN
      q <= b;
   ELSE
      q <= c;
   END IF;
END PROCESS;
Case Statement

- **Format:**

```
CASE {expression} IS
    WHEN <condition1> =>
        {sequence of statements}
    WHEN <condition2> =>
        {sequence of statements}
    WHEN OTHERS => -- (optional)
        {sequence of statements}
END CASE;
```

- **Example:**

```
PROCESS (sel, a, b, c, d)
BEGIN
    CASE sel IS
        WHEN "00" =>
            q <= a;
        WHEN "01" =>
            q <= b;
        WHEN "10" =>
            q <= c;
        WHEN OTHERS =>
            q <= d;
    END CASE;
END PROCESS;
```
Sequential LOOPS

- Infinite Loop
  - Loops infinitely unless EXIT statement exists

- While Loop
  - Conditional test to end loop

- FOR Loop
  - Iteration Loop

```
[loop_label]LOOP
  --sequential statement
  EXIT loop_label ;
END LOOP;
```

```
WHILE <condition> LOOP
  --sequential statements
END LOOP;
```

```
FOR <identifier> IN <range> LOOP
  --sequential statements
END LOOP;
```
Two Types of Process Statements

- **Combinatorial Process**
  - Sensitive to all inputs used in the combinatorial logic

**Example**

```
PROCESS(a, b, sel)
```

- **Sequential Process**
  - Sensitive to a clock or/and control signals

**Example**

```
PROCESS(clr, clk)
```

`sensitivity list includes all inputs used in the combinatorial logic`

`sensitivity list does not include the d input, only the clock or/and control signals`
DFF - rising_edge

---

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff_b IS
PORT ( d : in std_logic;
       clk : in std_logic;
       q : out std_logic
    );
END dff_b;

ARCHITECTURE behavior OF dff_b IS
BEGIN
PROCESS(clk)
BEGIN
IF rising_edge(clk) THEN
q <= d;
END IF;
END PROCESS;
END behavior;

---

rising_edge
- IEEE function that is defined in the std_logic_1164 package
- specifies that the signal value must be 0 to 1
- X, Z to 1 transition is not allowed
DFF with asynchronous clear

This is how to implement asynchronous control signals for the register.

Note: This IF-THEN statement is outside the IF-THEN statement that checks the condition `rising_edge`

Therefore, `clr='1'` does not depend on the clock.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY dff_clr IS
PORT ( clr : in bit;
    d, clk : in std_logic;
    q : out std_logic
); END dff_clr;

ARCHITECTURE behavior OF dff_clr IS
BEGIN
PROCESS(clk, clr)
BEGIN
    IF clr = '0' THEN
        q <= '0';
    ELSIF rising_edge(clk) THEN
        q <= d;
    END IF;
END PROCESS;
END behavior;
```
Design Hierarchically - Multiple Design Files

- VHDL hierarchical design requires Component Declarations and Component Instantiations

```
-- top.vhd
entity-architecture "top"
component "mid_a"
component "mid_b"

-- mid_a.vhd
entity-architecture "mid_a"
component "bottom_a"

-- mid_b.vhd
entity-architecture "mid_b"
component "bottom_a"
component "bottom_b"

-- bottom_a.vhd
entity-architecture "bottom_a"

-- bottom_b.vhd
entity-architecture "bottom_b"
```
Component Declaration and Instantiation

Component Declaration - Used to declare the Port types and the Data Types of the ports for a lower-level design

```component
COMPONENT <lower-level_design_name> IS
    PORT ( <port_name> : <port_type> <data_type>;
        ...
        <port_name> : <port_type> <data_type>);
END COMPONENT;
```

Component Instantiation - Used to map the ports of a lower-level design to that of the current-level design

```instantiation
<instance_name> : <lower-level_design_name>
    PORT MAP(<lower-level_port_name> => <current_level_port_name>,
        ...,<lower-level_port_name> => <current_level_port_name>);```
Library Altera/LPM

- LIBRARY ALTERA ;
  - Contains the following packages:
    - maxplus2 (Component declarations for all primitives and old-style megafunction Altera libraries)
    - megacore (Component declarations for some Altera Megacores)

- LIBRARY LPM;
  - Contains the following packages:
    - lpm_components (Component Declarations for all Altera LPM functions)

⇒ Note: See MAX+PLUS II or Quartus online help for more information
LPM Instantiation

- All of the Altera LPM macrofunctions are declared in the package `lpm_components.all` in the `LIBRARY lpm`;
- The MegaWizard Plug-in Manager in MAX+plus II and Quartus creates the VHDL code instantiating the LPM or Megafunction;
- In the VHDL Code:

```vhdl
LIBRARY lpm;
USE lpm.lpm_components.all;
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY tst_mult IS
PORT ( a, b : in std_logic_vector(7 downto 0);
  q_out : out std_logic_vector(15 downto 0));
END tst_mult;

ARCHITECTURE behavior OF tst_mult IS
BEGIN
  u1 : lpm_mult GENERIC MAP (lpm_widtha => 8, lpm_widthb => 8,
    lpm_widths => 16, lpm_widthp => 16)
  PORT MAP(dataa => a, datab => b, result => q_out);
END behavior;
**Exemplo 10 - HelloWorld**

- Fazer um led piscar a uma frequência de 1 s
- Utiliza a frequência de 50 MHz
Exemplo 10 – HelloWorld - VHDL

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

entity HelloWorld is
  port ( 
    clk: in std_logic;
    led: out std_logic 
  );
end HelloWorld;
```
architecture rtl of HelloWorld is
constant CLK_FREQ : integer := 50000000;
constant BLINK_FREQ : integer := 2;
constant CNT_MAX : integer := CLK_FREQ/BLINK_FREQ/2-1;
-- Sinais
signal cnt : unsigned(24 downto 0);
signal blink : std_logic;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if cnt=CNT_MAX then
                cnt <= (others => '0');
                blink <= not blink;
            else
                cnt <= cnt + 1;
            end if;
        end if;
    end process;
    led <= blink;
end rtl;
Exemplo 10 – HelloWorld

<table>
<thead>
<tr>
<th>Node Name</th>
<th>Direction</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>clk</strong></td>
<td>Input</td>
<td>PIN_T1</td>
</tr>
<tr>
<td><strong>led</strong></td>
<td>Output</td>
<td>PIN_V2</td>
</tr>
<tr>
<td>&lt;&lt;new node&gt;&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Copyright © 2003 Altera Corporation
Exemplo 10 – HelloWorld - TestBench

- Elaboração de um TestBench para o HelloWord usando o Quartus II - University Program WVF
- Simulação máxima 100 us
- O VHDL precisa ser alterado para comportar esse tempo
- Sugere-se uma frequência de 500 kHz
Exemplo 10 – HelloWorld - TestBench
Exemplo 10 – HelloWorld - TestBench
Exemplo 10 – HelloWorld - TestBench
Exemplo 10 – HelloWorld - TestBench
Exemplo 10 – HelloWorld - TestBench