“RAPID SYSTEM PROTOTYPING OF ELECTRONIC SYSTEMS: A VHDL OVERVIEW”

By

Luis E. Navarrete, MSEE
Orlando J. Hernandez, Ph.D.
Presentation Overview

- Modern Electronic Design Overview
  - Digital Logic Technologies
  - Programmable Logic Devices - FPGAs
- Introduction to VHDL – Part I
- Introduction to VHDL – Part II
  - AND, OR, HALF ADDER, FULL ADDER
- Introduction to VHDL – Part III
  - ALU Design
- Control and Data Path Organization
  - Finite State Machines, Digital Filter
- Introduction to MAX+PLUS II
- Implementations and Examples on the ALTERA UP1 Board
- Q&A Sessions
Modern Electronic Design

- Electronic Circuit Design
  - Analog Design
  - Digital Design

- Analog & Digital \(\rightarrow\) Mixed Signal Design

- Software and Hardware Engineers Plays an Increasing Important Role in Embedded Systems
Digital Logic Technologies

- Standard Logic
  - TTL 74xx
  - CMOS 4xxx

- Programmable Logic Devices (PLDs)
  - PLA
  - PAL
  - CPLD
  - FPGA

- ASICs
  - Standard Cell
  - Gate Array
  - Configurable Standard Cells

- Full Custom
  - VLSI
  - Microproc. & RAMs

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
Digital Logic Technologies
Programmable Logic Devices PLD:

- Chips that contains relatively large amounts of logic circuitry with a structure that is not fixed. Introduced in the 1970’s. Several types of PLD’s have been available:
  - Programmable Logic Array (PLA)
  - Programmable Array Logic (PAL)
  - Complex Programmable Logic Devices (CPLDs)
  - Field Programmable Gate Arrays (FPGAs)
Digital Logic Technologies
Programmable Logic Array PLA:

- Based on the idea that logic functions can be realized in sum-of-products form.
- A PLA comprises a programmable collection of AND gates that feeds a set of OR gates, configured to realize any sum-of-product functions of the PLA.

Inputs
Digital Logic Technologies
Programmable Logic Array PLA:

- PLA, Schematic:
Digital Logic Technologies
Programmable Array Logic PAL:

- In a PLA both the AND and the OR planes are programmable.
- Historically the programmable switches presents two difficulties for the manufacturers, they are hard to fabricate and reduced speed-performance.
Digital Logic Technologies
Programmable Array Logic PAL:

These Drawbacks led to the development of a device were only the AND Plane is programmable while the OR plane remains fixed. This devices were called the PALs.
Digital Logic Technologies
Programmable Array Logic PAL:

PAL, Schematic:
Digital Logic Technologies
Complex Programmable Logic Devices CPLDs:

- CPLDs were created as a substitute for the PLAs and PALs which are useful for implementing a wide variety of small digital circuits.

- This small digital circuits will be limited to a relatively small amount of inputs and outputs.
Digital Logic Technologies
Complex Programmable Logic Devices CPLDs:

- For implementing more complex circuits, either multiple PLAs or PALs can be used or a CPLD.

- A CPLD comprises multiple circuit blocks on a chip, with internal wiring resources to connect the circuit blocks. Each circuit block is similar to a PLA or a PAL.
Digital Logic Technologies

Complex Programmable Logic Devices (CPLDs):

- CPLDs, Schematics:
Digital Logic Technologies
Field Programmable Gate Arrays FPGAs:

- Is a programmable logic device that supports implementation of relatively large logic circuits (Much more than 20,000 gates).

- FPGAs, does not contains AND or OR planes as the CPLDs. Instead, FPGAs provide logic blocks for implementation of the required functions.
Digital Logic Technologies
Field Programmable Gate Arrays FPGAs:

- They contain three main types of resources:
  - Logic Blocks
  - I/O blocks for connecting to the package pins
  - and interconnection wires and switches

- More sophisticated state of the art FPGAs also contain:
  - Complex I/Os
  - Memories
  - Analog
  - Custom laid out processors
FPGAs, Schematics:
Digital Logic Technologies
Application Specific Integrated Circuit, ASICs:

- Chips that are design using state of the art VLSI technologies and that are tailored to some specific needs (or applications).

- In the 1980’s Application Specific Integrated Circuits (ASIC) designs were focused to meet time-to-market and customers specific requirements.
Digital Logic Technologies
Application Specific Integrated Circuit, ASICs:

- There are several integrated circuits design options:
  - Standard Cell Design
  - Gate Array
  (are among the main ones).
  - Emerging technologies are combining the density and cost efficiency of standard cells with the flexibility and quick design times of gate arrays: configurable standard cells
Digital Logic Technologies

Standard Cell Design:

- It is a design methodology where a library of macros (cells), which are predefined and pre-laid-out, is provided by a vendor.

- The user designs his/her circuit with these cells (logic function blocks) resulting in a schematic defining the interconnects among selected cells.
Typical libraries are created with low gate level primitives such as: AND, OR, NAND, NOR, XOR, Inverters, flip-flops, registers and other similar components.
Digital Logic Technologies

Gate Array Design:

- The Gate Array design uses a custom interconnection pattern of an array of uncommitted logic gates. These are called Gate Arrays.

- Wafers of chips containing the uncommitted logic gate arrays can be pre-fabricated up to the point of the final metallization steps which creates the logic personalization.
Digital Logic Technology Tradeoffs

Speed, Density, Complexity, Market Volume

Engineering Cost, Time to Develop Product
Field Programmable Logic Arrays, FPGAs

- FPGAs are Integrated Circuits Whose Internal Functional Operation is Defined by the User

- RAPID SYSTEM PROTOTYPING OF ELECTRONIC SYSTEMS.

- They can be Programmed using a Hardware description Language, VHDL
UP1 Development Board
www.altera.com
Session I

INTRODUCTION TO VHDL - PART I
Design Automation

- Need To Keep With Rapid Changes, Electronic Products Have To Be Designed Extremely Quickly
- Electronic Design Automation (EDA)
  - Design Entry
  - Simulation
  - Synthesis
  - Design Validation & Test
Design Automation. Cont...

- **Design Entry**
  - Schematic Capture
Design Entry - Textual Form:

- VHDL (*V*ery *H*igh *S*peed *I*ntegrated *C*ircuits)
- VHSIC (*V*ery *H*igh *S*peed *I*ntegrated *C*ircuits)
Design Entry - Textual Form:

```vhdl
entity and_2 is
  port (X, Y:in BIT; Z:out BIT);
end entity and_2;

architecture DATAFLOW of and_2 is
begin
  Z <= X and Y;
end architecture DATAFLOW;
```
Introduction To VHDL

- VHDL: Very High Speed Integrated Circuit (VHSIC) Hardware Description Language.

- VHDL Is an Industry Standard Language to Describe Hardware From the Abstract to Concrete Level.
BRIEF HISTORY OF VHDL

- VHDL is a Derivative of the VHSIC Program by US Dept. of Defense Along With IBM, Texas Instruments, and Intermetrics.

- In 1986 VHDL Became IEEE Standard and After Several Revisions. It Was Adopted As the IEEE 1076 Standard.
BRIEF HISTORY OF VHDL

- In 1999 the Analog and Mixed Signal Extensions Were Added to VHDL (VHDL-AMS)

- In 2000 the latest upgrade (Object Oriented added…)

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
MOTIVATION

- Need a Method to Quickly Design, Implement, Test and Document Increasingly Complex Digital Systems.

- Schematic and Boolean Equations Inadequate for Million-Gate ICs.

- Design Portability
What is VHDL?

- A Design entry language
- A Simulation modeling language.
- A Verification language.
- A Standard language.
- As simple or complex as required.
How is VHDL Used?

- For Design Specification ("Specify")
- For Design Entry ("Capture")
- For Design Simulation ("Verify")
- For Design Documentation ("formalize")
- As an Alternate to Schematics
FPGA Design Process

- VHDL Can Be Used for Both Design and Test Development

Design Entry → Test Development

Synthesis → Functional Simulation

Device Mapping → Timing Simulation

Device
When Should VHDL Be Used?

- VHDL is highly beneficial to use as a structured, top down approach to design.
- VHDL makes it easy to build, use, and reuse libraries of circuit elements.
- VHDL can greatly improve your chances of moving into more advanced tools and device targets.
Advantages of VHDL

- The Ability to Code the Behavior and to Synthesize an Actual Circuit.

- Power and Flexibility

- Device (specific FPGA) Independent Design
Advantages of VHDL Cont...

- Portability Among Tools and Devices
- Fast Switch Level Simulations
- Quick Time to Market and Low Cost
- Industry Standard
Getting Started with VHDL

- Its Easy To Get Started With VHDL, But Its Difficult To Master It.
- To Begin With, A Subset of The Language Can Be Learned To Write Useful Models.
- Later More Complex Features Can Be Learned To Implement Complex Circuits.
A First look at VHDL

- Lets start with a simple Combinational circuit: an 8-bit Comparator.
An 8 Bit Comparator

Comparator Specifications:

- Two 8-bit inputs
- 1-bit Output
- Output is 1 if the inputs match or 0 if they differ.
An 8 Bit Comparator

 Comparator

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL
--- Eight-bit Comparator

**entity** compare **is**

**port** (A, B : **in** std_logic_vector (0 to 7);

EQ : **out** std_logic);

**end** compare;

**architecture** comp **of** compare **is**

**begin**

EQ <= '1' when (A=B) **else** '0';

**end** comp;

- An entity declaration that defines the inputs and outputs - the ports of the circuit
- An architecture that defines the function of the circuit
Entities and Architectures

- Every VHDL design description has at least one entity/architecture pair.

- A large design has many entity/architecture pairs and are connected to form the complete circuit.
What is an Entity?

- An entity declaration describes the circuit as it appears from “outside”- from perspective of its input and output interfaces.

- An entity declaration is analogous to a block symbol on a schematic.
What is an Entity?

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

- The entity declarations includes a name, compare, and a port statement defining all the inputs and outputs of the entity.
What is an Architecture?

- Architecture Describes the Actual Function - or Contents of the Entity to Which It Is Bound.

- Architecture Is Roughly Analogous to a Lower Level Schematic Referenced by the High Level Functional Block Symbol.
What is an Architecture?

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

- The architecture declaration begins with a unique name, `comp`, followed by the name of the entity to which the architecture is bound, in this case `compare`. 
What is an Architecture?

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

- Between the keywords `begin` and `end` is found the actual functional description of the comparator.
Data Types

- VHDL’s high level data types allow data to be represented in much the same way as in high-level programming languages.

- A data type is an abstract representation of stored data.
Data Types

- These data types might represent individual wires in a circuit, or a collection of wires.
Data Types
## Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Values</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>'1', '0'</td>
<td>Q &lt;= '1';</td>
</tr>
<tr>
<td>Bit_vector</td>
<td>(array of bits)</td>
<td>Data &lt;= &quot;101001&quot;</td>
</tr>
<tr>
<td>Boolean</td>
<td>True, False</td>
<td>EQ &lt;= True;</td>
</tr>
<tr>
<td>Integer</td>
<td>-2, -1, 0, 1, 2, 3</td>
<td>C &lt;= c+2;</td>
</tr>
<tr>
<td>Real</td>
<td>1.0, -1.0E5</td>
<td>V1 = V2/5.3;</td>
</tr>
<tr>
<td>Time</td>
<td>1ua, 100ps</td>
<td>Q &lt;= '1' after 6 nS</td>
</tr>
<tr>
<td>Character</td>
<td>'a', 'b', '2', '$'</td>
<td>Char &lt;= 'X';</td>
</tr>
<tr>
<td>String</td>
<td>(Array of characters)</td>
<td>Msg &lt;= &quot;MEM&quot;;</td>
</tr>
</tbody>
</table>
Design Units

- Design units are a concept unique to VHDL that provide advanced configuration management capabilities.

- Design units are segments of VHDL code that can compiled separately and stored in a library.
Design Units

- Configuration (or default configuration)
- Package
- Package Body
- Entity
- Architecture (s)

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
A Package is a collection of commonly used data types to be used globally among different design units.

Package declaration is identified by the `package` keyword.
Package Design Unit

- Items defined within a package can be made visible to any other design unit in the complete VHDL design and they can be compiled into libraries for later re-use.

- A package can consist of two basic parts
  - A package declaration
  - A package body (optional)
Package Design unit

- Syntax of package

```
package my_package is
  function my_global_function ( .......... ) return bit;
end my_package;
```
Package Body

- The package body defines the actual behavior of the items specified in the package.

- The relationship between package and package body is somewhat similar to that of entity and its corresponding architecture.
VHDL Configurations

- VHDL allows you to create more than one alternate architecture for an entity.

- This feature helps in experimenting with different implementations of circuit description.

- It's also useful for simulation and for project team environments.
VHDL Configurations

- Configuration declarations are not generally used for synthesis, and may not be supported by the synthesis tools.

- Configuration declarations are always optional, even for complex circuits.
VHDL Configurations

An example of configuration declaration

configuration t_build of rcomp is
  for structure
    for COMP1: compare use entity
      work.compare (comp);
    for ROT1: rotate use entity
      work.rotate (rotate1);
  end for;
end t_build;
### Design Description

#### More Typical Design Description

<table>
<thead>
<tr>
<th>Standard Libraries</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Libraries</td>
<td>Package</td>
</tr>
<tr>
<td>Entity/ Architecture</td>
<td>Entity/ Architecture</td>
</tr>
<tr>
<td>Entity/ Architecture</td>
<td>Entity/ Architecture</td>
</tr>
<tr>
<td></td>
<td>Configuration</td>
</tr>
</tbody>
</table>
Levels of Abstraction (Styles)

- VHDL supports many possible styles of design description.

- These styles differ primarily in how closely they relate to the underlying hardware.
Levels of Abstraction (Styles)

- Levels of Abstraction refers to how far your design description is from an actual hardware realization.

- The three main levels of abstraction are:
  - Behavior
  - Dataflow (RTL)
  - Structure
Levels of Abstraction (Styles)

- **Behavior**
  - Performance Specification
  - Test Benches
  - Sequential Description
  - State Machines
  - Register Transfers
  - Selected Assignments
  - Arithmetic Operation
  - Boolean Equations
  - Hierarchy
  - Physical Information

- **Dataflow**

- **Structure**
Behavioral Modeling

- The Highest Level of Abstraction Supported in VHDL.

- The Behavior Approach Describes the Actual Behavior of Signals Inside the Component.
The Concept of Time Is the Critical Distinction Between Behavioral Descriptions and Low Level Descriptions.

The Concept to Time May Be Expressed Precisely, With Actual Delays Between Related Events.
An Example of Behavioral Modeling: A half adder

a

b

sum

carry
Half_adder

- Inputs a, b : 1 bit each.
- Output Sum, Carry : 1 bit each.

![Half adder circuit](image)

Figure 1-1 Half adder circuit
VHDL Code for half_adder

-- Half Adder
library IEEE; -- refer IEEE library.
use IEEE.std_logic_1164.all;

entity half_adder is
  port (a, b : in std_logic; -- declaring I/O ports
        sum, carry : out std_logic);
end half_adder;
half_adder code (cont...)

architecture behavior of
  half_adder is
begin
  sum  <= (a xor b);
  carry <= (a and b);
end behavior;

Figure 1-1 Half adder circuit
Max+Plus II Development Software

- MAX+PLUS II is a Development software created by ALTERA
- This software supports schematic capture and text-based hardware description language (HDL) design entry, including Verilog® HDL, VHDL, and the Altera® Hardware Description Language (AHDLTM)
Max+Plus II
Development Software

- It also provides design programming, compilation, and verification support for all devices supported by the MAX+PLUS II BASELINE software including the EPM7128S, EPF10K20, and EPF10K70 devices.
Max+Plus II
Development Software

- The MAX+PLUS II University software can be freely distributed to students for installation on their personal computers and provides instant access to online help.

- For more information, follow the University Program Link at www.altera.com
Max+Plus II
Development Software

This Tool was created to help during the various implementation steps:

- Design
  - Graphical Entry
  - VHDL Model
- Compilation
- Simulation
- Verification
- Synthesis
Max+Plus II
1st Part of The Process !!!!

**Design**

Graphical Entry

HDL Model

**Compilation**

Compiler

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003

UNIVERSIDAD DEL NORTE
Max+Plus II
2nd Part of The Process !!!!

Compilation

Simulation

Timing Diagram

Timing Analysis

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
Max+Plus II
3rd Part of The Process !!!!

Compilation

Verification

Programming

Compiler

UP1 Board

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003

UNIVERSIDAD DEL NORTE
Max+Plus II Development Software

- MAX+PLUS II University software targets the UP1 and UP1X development boards.

- The FLEX10K and MAX7000S devices can be configured and programmed (respectively) to interact with the included hardware or with any other external design.
Max+Plus II Development Software
Max+Plus II

Half-Adder

- We will use our previous Half-Adder model as an example of how to perform a VHDL code entry.
Max+Plus II
Half-Adder, Code Entry

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

entity half_adder is
  port(a, b : in std_logic;
       sum, c_out: out std_logic);
end entity half_adder;

architecture behavior of half_adder is
begin
  sum <= (a xor b);
  c_out <= (a and b);
end architecture behavior;
```
Max+Plus II
Half-Adder, Compiler
Max+Plus II
Half-Adder, Simulation
Session II

INTRODUCTION TO VHDL - PART II
The dataflow level of abstraction is often called **Register Transfer Language (RTL)**.

The dataflow level of abstraction describes how information is passed between registers in the circuit.
Concurrent and Sequential VHDL

- VHDL Allows Both Concurrent and Sequential Statements to Be Entered.

- The Difference Between Concurrent and Sequential Statements Must Be Known for Effective Use of the Language.
Concurrent VHDL

- All Statements in the Concurrent Area Are Executed at the Same Time.

- There Is No Significance to the Order in Which Concurrent Statements Occur.
Concurrent VHDL

```
Begin
    Statement
    Statement
    Statement
End
```
Example of Concurrent VHDL

Full Adder

a

Full-Adder

b

Sum

C_in

C_out
Full Adder circuit

\[
\begin{align*}
&\text{a} \\
&\text{b} \\
&\text{c}_{\text{in}} \\
&\text{s}_1 \\
&\text{s}_3 \\
&\text{s}_2 \\
&\text{sum} \\
&\text{c}_{\text{out}}
\end{align*}
\]
VHDL code for Full Adder

```vhdl
-- Full_Adder
library IEEE;
use IEEE.std_logic_1164.all;

entity full_adder is
port (a, b, c_in : in std_logic;
    sum, c_out : out std_logic);
end full_adder;
```
VHDL code for Full Adder

architecture dataflow of full_adder is
signal s1, s2, s3 : std_logic;
begin
  L1: s1 <= (a xor b);
  L2: s2 <= (c_in and s1);
  L3: s3 <= (a and b);
  L4: sum <= (s1 xor c_in);
  L5: c_out <= (s2 or s3);
end dataflow;

-- Using Signal Assignment Instructions
Next we will use MAX+PLUS II to Compile and Simulate our previous Full-Adder model.
Max+Plus II
Full-Adder, Code Entry

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

datatype full_adder is
port(a, b, c_in: in std_logic;
    sum, c_out: out std_logic);
end datatype full_adder;

datatype dataflow of full_adder is
signal s1, s2, s3: std_logic;
begin
    L1: s1 <= (a xor b);
    L2: s2 <= (c_in and s1);
    L3: s3 <= (a and b);
    L4: sum <= (s1 xor c_in);
    L5: c_out <=(s2 or s3);
end datatype dataflow;
```
Max+Plus II
Full-Adder, Compiler
Max+Plus II
Full-Adder, Simulation

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
VHDL code for Full Adder

The expressions labeled L1-L5 are all concurrent signal assignment statements. All the statements are executed at the same time.

L1: \( s_1 \leq (a \ xor \ b) \);
L2: \( s_2 \leq (c_{\text{in}} \ and \ s_1) \);
L3: \( s_3 \leq (a \ and \ b) \);
L4: \( \text{sum} \leq (s_1 \ xor \ c_{\text{in}}) \);
L5: \( c_{\text{out}} \leq (s_2 \ or \ s_3) \);
VHDL code for Full Adder

- The simulator evaluates all the expressions, L1-L5, then applies the results to the signals.

- Once the simulator has applied the results it waits for one of the signal to change and it reevaluates all the expression again.
VHDL code for Full Adder

- This cycle will continue until the simulation is completed.
- This is called “event driven simulation”.
- It is more computationally efficient than time driven simulation.
Signals

- In the Full_adder VHDL code we came across “signal”.
  
  ```vhdl
  architecture dataflow of full_adder is
  signal s1, s2, s3: std_logic;
  begin
  .......
  end dataflow;
  ```

- So what are “signals”?
Signals

- Signals Are Used to Carry Data From Place to Place in a VHDL Design Description.

- Signals Are Similar to Wires in a Schematic.

- Signals are internal to an entity, so they exist only in the architectures.
Sequential VHDL

- Sequential Statements Are Executed One After the Other in the Order That They Appear.

- Example of Sequential Statement: Process.
Sequential VHDL

Begin

Statement

Statement

Statement

End
Process Construct

- The Process construct is the primary means to describe sequential operations.

- Process starts with the keyword `process` and ends with the keyword `end process`.

- The `process` construct itself is treated as a concurrent statement.
Process Statement

- The *process* consists of three parts:
  - Sensitivity List
  - Process declaration part
  - Statement Part
Syntax of Process Statement

```
architecture arch_name of ent_name is
begin
  process_name: process (sensitivity_list)
  local_declaration;

  .......

  begin
  sequential statement;
  sequential statement;

  ....

  end process;

end arch_name;
```
library IEEE;
use IEEE.std_logic_1164.all;
entity nand2 is
  port (a, b : in std_logic;
    c : out std_logic);
end nand2;
architecture arch1 of nand2 is
begin
  process (a, b)
  variable temp: std_logic;
  Process Example
Process Example

begin
  temp = not (a and b);
  if (temp = '1') then
    c <= temp after 5 ns;
  elsif (temp = '0') then
    c <= temp after 6 ns;
  end if;
end process;
end arch;
library IEEE;
use IEEE.std_logic_1164.all;
entity nand2 is
    port( a, b : in std_logic; c: out std_logic);
end entity nand2;
architecture arch1 of nand2 is
begin
    process( a, b )
        variable temp: std_logic;
    begin
        temp := not ( a and b);
        if (temp = '1') then c <= temp after 5 ns;
        elsif (temp = '0') then c <= temp after 6 ns;
        end if;
    end process;
end architecture arch1;
Max+Plus II
Nand2, Simulation
Example Description

- library and use statements allowing to use the IEEE 1164 standard logic data types.

```
library IEEE;
use IEEE.std_logic.all
```
Example Description

- An **entity** declaration defining the interface to the circuit.

```vhdl
entity nand2 is
  port (a, b : in std_logic;
       c : out std_logic);
end nand2;
```
Example Description

- The **process** sensitivity list enumerates exactly which signals causes the process execution.

```
process (a, b)
```
The process declarative part is used to declare local variables or constants that can be used in the process.

```
variable temp: std_logic
```
Example Description

- Variables are temporary storage areas similar to variables in software programming languages.

```
variable temp: std_logic
```
Use of Sequential Statements

Sequential Statements Exist Inside the Process Statements As Well As in Sub Programs.

The Sequential Statements Are:

- if
- case
- loop
- assert
- wait
IF Statements

- The IF statement starts with the keyword `if` and ends with the keyword `end if`.

```
If (x < 10) then
  a := b;
end if;
```
IF Statements

- There are also two optional clauses
  - Elsif clause
  - Else clause

```
if (day = Sunday) then
  weekend := true;
elsif (day = Saturday) then
  weekend := true;
else
  weekday := true;
end if;
```

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL
The **if** statement can have multiple **elsif** statement parts but only one else statement part.
The Case statement is used whenever a single expression value can be used to select between a number of actions.

A Case statement consists of the keyword `case` followed by an expression and the keyword `is`.
Case statement

- The expression will either return a value that matches one of the *choices* in a *when* statement part or match an *others* clause.
type vectype is array(0 to 1) of bit;
variable bit_vec: vectype;

......
case bit_vec is
  when "00" =>
    return 0;
  when "01" =>
    return 1;
  when "10" =>
    return 2;
  when "11" =>
    return 3;
end case;
Loop Statements

- The loop statement is used whenever an operation needs to be repeated.

- Loop statements are implemented in two ways
  - `while` condition loop statement
  - `for` condition loop statement
Loop Statements (while)

- The **while** condition Loop statement will loop as long as the condition expression is TRUE.

```vhdl
while (day = weekday) loop
  day := get_next_day (day);
end loop;
```
Loop Statements (for loop)

```
for i in 1 to 10 loop
  i_squared(i) := i*i;
end loop;
```

This loop will execute 10 times whenever execution begins and its function is to calculate squares from 1 to 10 and insert them into i_squared signal array.
Next statement

The **next** statement allows us to stop execution of a particular iteration and go on to the next iteration.

```vhdl
for i in 0 to max_limit loop
    if (done (i) = true ) then
        next;
    else
        done(i) := true;
    end if;
    q(i) <= a(i) and b(i);
end loop;
```
Exit statement

The VHDL exit statement allows the designer to exit or jump out of the loop statement currently in execution.

```vhdl
for i in 0 to max_limit loop
  if (int_a <= 0) then
    exit;
  else
    int_a := int_a-1;
    q(i) <= 3.14 / real(int_a * i);
  end if;
end loop;
```

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL
Assert Statements Are Very Useful for Reporting Textual Strings to the Designer.

The Assert Statement Checks the Value of a Boolean Expression. If the Condition Is Not True, a Report (Message) Is Generated During the Simulation.
Assert Statement

An assert statement includes two options, either or both of which may be used. The first Option

- **Report:** displays a user defined message if the condition is false
Assert Statement

An assert statement includes two options, either or both of which may be used. The second Option:

- **Severity**: allows the user to choose a severity level if the condition is false. The four levels of severity are
  - Note,
  - Warning,
  - Error
  - Failure
Wait Statements

- The *wait* statement allows to suspend the sequential execution of a process or subprogram.

- The condition for resuming execution of the suspended process or subprogram can be specified by three different means.
### Wait Statements

- **wait on** signal changes.

  ```
  wait on signal [signal]
  ```

- **wait until** an expression is true.

  ```
  wait until Boolean_expression
  ```
Wait Statements

- **wait for** a specific amount time.

```
wait for time_expression
```
The *wait on* signal clause specifies a list of one or more signals upon which the *wait* statement will wait for the events.

```vhdl
process
begin
  if (reset = '1') then
    q <= '0';
  elsif clock 'event and clock = '1' then
    q <= d;
  end if;
  wait on reset, clock;
end process;
```

* Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL
* Universidad del Norte
* May 21 - 23, 2003
* 139
Wait Until Statement

- The **wait until** Boolean_expression clause will suspend execution of the process until the expression returns a true value.

```vhdl
process
begin
  wait until clock = '1' and clock' event;
  q <= d ;
end process;
```
Wait for Statement

- The `wait for` time_expression clause will suspend execution of the process for the time specified by the time expression.

```vhdl
wait for 10 ns;
```

- The wait statement will suspend for 10 ns and after 10 ns the execution will continue.
Structural VHDL

- Structural-level design methods can be useful for managing the complexity of a large design description.

- Structure level of abstraction is used to combine multiple components to form a larger circuit.
Structural VHDL Descriptions Are Quite Similar in Format to Schematic Netlists.

Larger Circuits Can Be Constructed From Smaller Building Blocks.
Example of Structural VHDL

Let us consider an ALU with

- An OR gate
- An XOR gate
- A Half Adder
- A Full Adder
- A Multiplexer
Example of Structural VHDL

- OR gate
- XOR gate
- Half Adder
- Full Adder
- Mux
ALU – Block Diagram

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
146
ALU – Function Table

<table>
<thead>
<tr>
<th>S1</th>
<th>S0</th>
<th>Z</th>
<th>C_out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>a or b</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>a xor b</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>ha_sum</td>
<td>ha_c_out</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>fa_sum</td>
<td>fa_c_out</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s1</th>
<th>s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
</tbody>
</table>

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL
library IEEE;
use IEEE.std_logic_1164.all;
entity t_or is
  port (a, b : in std_logic;
        ored : out std_logic);
end t_or;
architecture concurrent_behavior of t_or is
begin
  ored <= (a or b);
end concurrent_behavior;
library IEEE;
use IEEE.std_logic_1164.all;

entity t_or is
  port(a,b : in std_logic; ored : out std_logic);
end entity t_or;

architecture concurrent_behavior of t_or is
begin
  ored<=(a or b);
end architecture concurrent_behavior;
Max+Plus II
T_OR, Compiler
Max+Plus II
T_OR, Simulation

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
library IEEE;
use IEEE.std_logic_1164.all;
entity t_xor is
  port (a,b : in std_logic;
       xored : out std_logic);
end t_xor;
architecture concurrent_behavior of t_xor is
begin
  xored <= (a xor b);
end concurrent_behavior;
Max+Plus II
T_XOR, Code Entry

library IEEE;
use IEEE.std_logic_1164.all;

entity t_xor is
  port(a,b : in std_logic; xored : out std_logic);
end entity t_xor;

architecture concurrent_behavior of t_xor is
begin
  xored<=(a xor b);
end architecture concurrent_behavior;
Max+Plus II
T_XOR, Compiler

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
Max+Plus II
T_XOR, Simulation
VHDL Code for Half_adder

-- Half Adder
library IEEE;                          --refer IEEE library.
use IEEE.std_logic_1164.all;

entity half_adder is
    port (a, b : in std_logic;
          sum, c_out : out std_logic);
end half_adder;

--declaring I/ O ports
architecture behavior of
half_adder is
begin
  sum <= (a xor b);
  c_out <= (a and b);
end behavior;

Figure 1-1 Half adder circuit
Max+Plus II
Half-Adder, Code Entry

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

entity half_adder is
  port(a, b : in std_logic;
       sum, c_out: out std_logic);
end entity half_adder;

architecture behavior of half_adder is
begin
  sum <= (a xor b);
  c_out <= (a and b);
end architecture behavior;
```
Max+Plus II
Half-Adder, Compiler
Max+Plus II
Half-Adder, Simulation

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en
Procesamiento Digital de Señales y Diseño Digital con VHDL
Full Adder circuit
VHDL code for Full Adder

```vhdl
--Full_Adder
library IEEE;
use IEEE.std_logic_1164.all;

entity full_adder is
  port (a, b, c_in : in std_logic;
        sum, c_out : out std_logic);
end full_adder;
```
architecture dataflow of full_adder is
signal s1, s2, s3 : std_logic;
begin
    L1: s1 <= (a xor b);
    L2: s2 <= (c_in and s1);
    L3: s3 <= (a and b);
    L4: sum <= (s1 xor c_in);
    L5: c_out <= (s2 or s3);
end dataflow;

-- Using Signal Assignment Instructions
Max+Plus II

Full-Adder, Code Entry

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

entity full_adder is
port(a, b, c_in: in std_logic;
    sum, c_out: out std_logic);
end entity full_adder;

architecture dataflow of full_adder is
signal s1, s2, s3: std_logic;
begin
    L1: s1 <= (a xor b);
    L2: s2 <= (c_in and s1);
    L3: s3 <= (a and b);
    L4: sum <= (s1 xor c_in);
    L5: c_out <=(s2 or s3);
end architecture dataflow;
```
Max+Plus II
Full-Adder, Compiler
Max+Plus II
Full-Adder, Simulation

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en
Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
library IEEE;
use IEEE.std_logic_1164.all;
package alu_pack is
  component t_or
    port (a, b : in std_logic;
          ored : out std_logic);
  end component;
  component t_xor
    port (a, b : in std_logic;
         xored : out std_logic);
  end component;
end package;
Declaration of ALU Package

```vhdl
component half_adder
  port (a, b : in std_logic;
       ha_sum, c_out : out std_logic);
end component;

component full_adder
  port (a, b, c_in : in std_logic;
       sum, c_out : out std_logic);
end component;

del alu_pack;
```

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL
library IEEE;
use IEEE.std_logic_1164.all;

package alu_pack is
  component t_or
    port(a,b:in std_logic; ored: out std_logic);
  end component;
  component t_xor
    port(a,b:in std_logic; xored: out std_logic);
  end component;
  component half_adder
    port(a,b:in std_logic; sum, c_out: out std_logic);
  end component;
  component full_adder
    port(a,b,c_in: in std_logic; sum,c_out: out std_logic);
  end component;
end alu_pack;
Max+Plus II
alu_pack, Compiler
The ALU_PACK design can not be simulated since it is a Package.

Its operation will be verified together with the designs that will use it. In this case this will be the ALU main design.
Main Code for ALU

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
library work;
use work.alu_pack.all;

entity alu is
  port (a, b, c_in, s0, s1 : in std_logic;
       z, c_out : out std_logic);
end alu;
```
Main Code for ALU Cont...

```vhdl
architecture structural of alu is

signal ored, xored, ha_sum, ha_c_out, fa_sum,
    fa_c_out : std_logic;

begin
    a1: t_or port map (a => a, b => b, ored => ored);
    x1: t_xor port map (a => a, b => b, xored => xored);
    h1: half_adder port map (a => a, b => b,
                              sum => ha_sum, c_out => ha_c_out);
    f1: full_adder port map (a => a, b => b,
                              c_in => c_in, sum => fa_sum,
                              c_out => fa_c_out);

    Main Code for ALU Cont...
```

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en
Procesamiento Digital de Señales y Diseño Digital con VHDL
Main Code for ALU Cont....

alu_process: process (a, b, c_in, s0, s1) begin

if (s1 = '0' and s0 = '0') then
    z <= ored;
    c_out <= '0';
end if;

if (s1 = '0' and s0 = '1') then
    z <= xored;
    c_out <= '0';
end if;
Main Code for ALU Cont....

```
if (s1 = '1' and s0 = '0') then
    z <= ha_sum;
    c_out <= ha_c_out;
end if;
if (s1 = '1' and s0 = '1') then
    z <= fa_sum;
    c_out <= fa_c_out;
end if;
end process alu_process;
end architecture structural;
```
Max+Plus II

ALU, Code Entry

library IEEE;
use IEEE.std_logic_1164.all;
library work;
use work.alu_pack.all;

entity alu is
  port(a, b, c_in, s0, s1: in std_logic;
       z, c_out: out std_logic);
end entity alu;

architecture structural of alu is
signal ored, xored, ha_sum, ha_c_out, fa_sum, fa_c_out :std_logic;
beginn
  a1:t_or port map (a=>a, b=>b, ored=>ored);
  x1:t_xor port map (a=>a, b=>b, xored=>xored);
  h1:half_adder port map (a=>a, b=>b, sum=>ha_sum, c_out=>ha_c_out);
  f1:full_adder port map (a=>a, b=>b, c_in=>c_in, sum=>fa_sum, c_out=>fa_c_out);

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en
Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
Max+Plus II
ALU, Compiler

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en
Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
Max+Plus II
ALU, Simulation

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003
UNIVERSIDAD DEL NORTE
Session III

CONTROL AND DATA PATH ORGANIZATION
Control and Data Path Organization

- Most complex digital circuits can be broken up into two parts:
  - Control
  - Data Path
Control and Data Path Organization

CONTROL INPUTS

CONTROL PROCESSING BLOCK

DATA PROCESSING BLOCK

DATA

CONTROL

OBSERVATION

STATUS
Finite State Machines

- Two Classes of Finite State Machines (FSMs):
  - Moore Machines
  - Mealy Machines
Moore Finite State Machines

- Outputs depend only on the state
- State and Outputs Processing are combinational elements
- State Vector is Sequential Elements
Mealy Finite State Machines

- Outputs depend on the state and the inputs
Session IV

VHDL IMPLEMENTATION EXAMPLES
- A Decimation Filter for a Sigma-Delta Analog to Digital Converter
2-Ch Σ-Δ Analog to Digital Converter
2nd Order Σ-Δ Modulator (block/algorithmic)

This can be modeled in Behavioral VHDL
2nd Order Σ-Δ Modulator (circuit)

This can also be modeled in Behavioral VHDL
Decimation Digital Filter

IN

\[ z^{-1} \]

\[ f_s \]

\[ \frac{f_s}{OSR} \]

OUT

\[ z^{-1} \]

\[ \frac{f_s}{OSR} \]
Decimation Digital Filter

- Cubic sinc
- Bits of noise free accuracy for delta-sigma ADC's:
  - BITS = 3 * LOG(OSR) / LOG(2) + 2
  - Assume OSR=32, then BITS=17, and set BITS=16
Decimation Digital Filter

- **First Filter Equations**
  - \( H_1(z) = \frac{Y_1(z)}{X(z)} = \frac{1}{1 - 3z^1 + 3z^2 - z^3} \)
  - \( y_1(n) = x(n) + 3y_1(n-1) - 3y_1(n-2) + y_1(n-3) \)

- **Second Filter Equations**
  - \( H_2(z) = \frac{Y(z)}{X_1(z)} = 1 - 3z^1 + 3z^2 - z^3 \)
  - \( y(n) = x_1(n) - 3x_1(n-1) + 3x_1(n-2) - x_1(n-3) \)

- **Decimation (Retiming)**
  - \( x_1(n) = y_1(n/OSR) \)
  - \( x_1(n) = y_1(n/32) \)
What do we need for our design?

- \[ y_1(n) = x(n) + 3 \cdot y_1(n-1) - 3 \cdot y_1(n-2) + y_1(n-3) \]
- \[ y(n) = x_1(n) - 3 \cdot x_1(n-1) + 3 \cdot x_1(n-2) - x_1(n-3) \]
- \[ x_1(n) = \frac{y_1(n)}{32} \]

**Control**
- On every \( x(n) \)
  - S02: Store \( x(n) \) in accumulator, count \( x(n) \) mod 32
  - S03: Accumulate 2 \( y_1(n-1) \)
  - S04: Accumulate \( y_1(n-1) \)
  - S05: Accumulate 1's complement of 2 \( y_1(n-2) \)
  - S06: Accumulate 1's complement of \( y_1(n-2) \)
  - S07: Accumulate 2
  - S08: Accumulate \( y_1(n-3) \)
  - S09: Update \( y \) registers
- On every \( x_1(n) \) (every 32\(^{nd} \) \( y_1(n) \))
  - S10: Accumulate 1's complement of 2 \( x_1(n-1) \)
  - S11: Accumulate 1's complement of \( x_1(n-1) \)
  - S12: Accumulate 2 \( x_1(n-2) \)
  - S13: Accumulate \( x_1(n-2) \)
  - S14: Accumulate 1's complement of \( x_1(n-3) \)
  - S15: Accumulate 3, output result
  - S16: Store \( y_1(n-1) \) in accumulator
  - S17: Update \( x \) registers

**Data Path**
- 16 bits
- Adder-Accumulator
- 1's complement
- Shift left by one (x 2)
- Store \( y_1(n-1), y_1(n-2), y_1(n-3) \)
- Store \( x_1(n-1), x_1(n-2), x_1(n-3) \)
- Constants: 2 & 3
Decimation Digital Filter Architecture

DATA-PATH

Ry1_n_1 → Ry1_n_2 → Ry1_n_3

Rx1_n_1 → Rx1_n_2 → Rx1_n_3

URy

URx

S1

S2

S3

1’S COMP., PASS, 2, 3

x2, PASS

xn

S4

ACCUMULATOR

OUT

CONTROLLER

S1

S2

S3

S4

URy

URx

OSS

ISS

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003

UNIVERSIDAD DEL NORTE

193
VHDL code for Data_Path

--Data_Path
library IEEE;
use IEEE.std_logic_1164.all;

entity Data_Path is
port (CLK, reset, xn, URy, URx, S2 : in std_logic;
     S3, S4 : in std_logic_vector(1 downto 0);
     S1 : in std_logic_vector(2 downto 0);
     OUTPUT : out std_logic_vector(15 downto 0));
end Data_Path;
architecture behave of Data_Path is
  signal Ry1_n_1, Ry1_n_2, Ry1_n_3 : std_logic_vector(15 downto 0);
  signal Rx1_n_1, Rx1_n_2, Rx1_n_3 : std_logic_vector(15 downto 0);
  signal ACCUMULATOR : std_logic_vector(15 downto 0);
begin
  output <= ACCUMULATOR;
  process (CLK, reset) begin
    constant my_zero : std_logic_vector(15 downto 0) := "0000000000000000";
    variable T1, T2, T3, T4, T5 : std_logic_vector(15 downto 0);
    variable my_msb : std_logic;
    if reset = '1' then
      Ry1_n_1 <= my_zero; Ry1_n_2 <= my_zero; Ry1_n_3 <= my_zero;
      Rx1_n_1 <= my_zero; Rx1_n_2 <= my_zero; Rx1_n_3 <= my_zero;
      ACCUMULATOR <= my_zero;
    elsif CLK'EVENT and CLK = '1' then
      if URy = '1' then
        Ry1_n_3 <= Ry1_n_2; Ry1_n_2 <= Ry1_n_1;
        Ry1_n_1 <= ACCUMULATOR;
      end if;
      if URx = '1' then
        Rx1_n_3 <= Rx1_n_2; Rx1_n_2 <= Rx1_n_1;
        Rx1_n_1 <= ACCUMULATOR;
      end if;
    case S1 is
    when "000" => T1 <= Ry1_n_1;
    when "001" => T1 <= Ry1_n_2;
    when "010" => T1 <= Ry1_n_3;
    when "011" => T1 <= Rx1_n_1;
    when "100" => T1 <= Rx1_n_2;
    when "101" => T1 <= Rx1_n_3;
    end case;
    case S2 is
    when '0' =>
      my_msb <= T1(15);
      T2 <= T1 slt 1;
      T3 <= T2 and "0111111111111111";
      T4 <= T3 or (my_msb & "0000000000000000");
    when '1' => T4 <= T1;
    end case;
    case S3 is
    when "00" => T5 <= not T4;
    when "01" => T5 <= T4;
    when "10" => T5 <= "0000000000000010";
    when "11" => T5 <= "0000000000000011";
    end case;
    case S4 is
    when "00" =>
      ACCUMULATOR <= "0000000000000000" & xn;
    when "01" => ACCUMULATOR <= Ry1_n_1;
    when "10" =>
      ACCUMULATOR <= ACCUMULATOR + T5;
    end case;
    end if;
  end case;
  end process;
end behave;
---Controller
library IEEE;
use IEEE.std_logic_1164.all;

entity Controller is
    generic (TPD : TIME := 1 nS);
    port (CLK, reset, ISS : in std_logic;
            URy, URx, S2, OSS : out std_logic;
            S3, S4 : out std_logic_vector(1 downto 0);
            S1 : out std_logic_vector(2 downto 0));
end Controller;
VHDL code for Controller

architecture Moore of Controller is
  type STATETYPE is (S00, S01, S02, S03, S04, S05, S06, S07, S08, S09, S10, S11, S12, S13, S14, S15, S16, S17);
  signal State : STATETYPE;
  signal Counter : std_logic_vector(4 downto 0);
begin
  URy <= '1' after TPD when State = S09 else '0' after TPD;
  URx <= '1' after TPD when State = S17 else '0' after TPD;
  with State select
  S1 <= "000" after TPD when S03 | S04 | S16, "001" after TPD when S05 | S06, "010" after TPD when S08, "011" after TPD when S10 | S11, "100" after TPD when S12 | S13, "101" after TPD when S14;
  S2 <= '0' after TPD when State = S03 or State = S05 or State = S10 or State = S12 else '1' after TPD;
  with State select
  S3 <= "00" after TPD when S05 | S06 | S10 | S11 | S14, "01" after TPD when S03 | S04 | S08 | S12 | S13 | S16, "10" after TPD when S07, "11" after TPD when S15;
  with State select
  S4 <= "00" after TPD when S02, "01" after TPD when S16, "10" after TPD when S03 | S04 | S05 | S06 | S07 | S08 | S10 | S11 | S12 | S13 | S14 | S15;
  OSS <= '1' after TPD when State = S15 else '0' after TPD;
VHDL code for Controller

```vhdl
process (CLK, reset) begin
    if reset = '1' then State <= S00;
    elsif CLK'EVENT and CLK = '1' then
        case State is
            when S00 =>
                if ISS = '1' then State <= S02;
                end if;
            when S01 =>
                if ISS = '1' then State <= S02;
                end if;
            when S02 =>
                if Counter = "11111" then
                    Counter <= "00000";
                else
                    Counter <= Counter + "00001";
                end if;
                State <= S03;
            when S03 => State <= S04;
            when S04 => State <= S05;
            when S05 => State <= S06;
            when S06 => State <= S07;
            when S07 => State <= S08;
            when S08 => State <= S09;
            when S09 =>
                if Counter = "00000" then
                    State <= S10;
                else
                    State <= S01;
                end if;
            when S10 => State <= S11;
            when S11 => State <= S12;
            when S12 => State <= S13;
            when S13 => State <= S14;
            when S14 => State <= S15;
            when S15 => State <= S16;
            when S16 => State <= S17;
            when S17 => State <= S01;
        end case;
    end if;
end process;
end Moore;
```

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003

UNIVERSIDAD DEL NORTE
Main Code for FILTER

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

entity FILTER is
    port (reset, CLK, ISS, xn : in std_logic;
          OSS : out std_logic;
          OUTPUT : out std_logic_vector(15 downto 0));
end entity FILTER;
```
Main Code for FILTER Cont…

architecture structural of FILTER is

signal URy, URx, S2 : std_logic;
signal S3, S4 : std_logic_vector(1 downto 0);
signal S1 : std_logic_vector(2 downto 0));

begin
  c: Controller port map (CLK => CLK, reset => reset,
                          ISS => ISS, URy => URy, URx => URx, S2 => S2,
                          OSS => OSS, S3 => S3, S4 => S4, S1 => S1);
  dp: Data_Path port map (CLK => CLK, reset => reset,
                          xn => xn, URy => URy, URx => URx, S2 => S2,
                          S3 => S3, S4 => S4, S1 => S1, OUTPUT => OUTPUT);

end architecture structural;
Conclusions

From Gates to Large IP

Tecnologías de la Información: Telecomunicaciones, Aplicaciones en Procesamiento Digital de Señales y Diseño Digital con VHDL

May 21 - 23, 2003

UNIVERSIDAD DEL NORTE
Conclusions
Implementations and Examples on the ALTERA UP1 Board
Thanks......

lenavarr@eng.usf.edu
hernande@a-wit.com