VHDL and Verilog – Module 1
Introduction

Jim Duckworth
ECE Department, WPI
Topics

• Background to VHDL
• Introduction to language
• Programmable Logic Devices
  – CPLDs and FPGAs
  – FPGA architecture
  – Nexys 2 Board
• Using VHDL to synthesize and implement a design
• Verilog overview
Hardware Description Languages

• Example HDL's: ABEL, VERILOG, VHDL

• Advantages:
  – Documentation
  – Flexibility (easier to make design changes or mods)
  – Portability (if HDL is standard)
  – One language for modeling, simulation (test benches), and synthesis
  – Let synthesis worry about gate generation
  • Engineer productivity

• However: A different way of approaching design
  – engineers are used to thinking and designing using graphics (schematics) instead of text.
VHDL

• **VHSIC Hardware Description Language**
  – Very High Speed Integrated Circuit
• Standard language used to describe digital hardware devices, systems and components
  – Developed initially for documentation
• VHDL program was an offshoot of the US Government's VHSIC Program
• Approved as an IEEE Standard in December 1987 (IEEE standard 1076-1987)
  – Revised - now 1076-1993 (supported by all tools)
  – Latest version VHDL-2008 (not yet supported by tools!)
    • Integration of 1164 std
    • General improvements, etc
VHDL References

• “FPGA Prototyping by VHDL Examples – Xilinx Spartan-3 Version, by Pong P. Chu, 2008, Wiley
• “RTL Hardware Design using VHDL – Coding for Efficiency, Portability, and Scalability” by Pong P. Chu, 2008, Wiley.
• “VHDL Made Easy” by David Pellerin and Douglas Taylor, 1997, Prentice Hall
What exactly is VHDL?

- A way of describing the operation of a system.
  - Example: a 2-input multiplexer

```vhdl
ENTITY mux IS
  PORT (a, b, sel : IN std_logic;
       y : OUT std_logic);
END mux;

ARCHITECTURE behavior OF mux IS
BEGIN
  y <= a WHEN sel = '0' ELSE b;
END behavior;
```
Example Synthesis Results (not Xilinx)
Basic Terminology

• Note: VHDL is case insensitive, free format.
• Comments are preceded by two consecutive dashes.
  – comment ends at end of current line
• A digital component is described using an
  – ENTITY DECLARATION and a corresponding
  – ARCHITECTURE BODY.
• Std_logic is an enumeration type defined in an IEEE package
  – has the values '0' and ‘1’ and ‘Z’ (and others)
• Ports are like IC pins, connected by wires called SIGNALS
IEEE STANDARD 1164

- Provides a standard data type (std_logic) - nine values
  - U uninitialized
  - X forcing unknown
  - 0 forcing logic 0
  - 1 forcing logic 1
  - Z high impedance
  - W weak unknown
  - L weak logic 0
  - H weak logic 1
  - - don’t care

- To use standard logic data types place at top of source file
  - LIBRARY ieee;
  - USE ieee.std_logic_1164.ALL;
Entity

- The ENTITY defines the external view of the component
- PORTS are the communication links between entities or connections to the device pins
- Note the use of libraries before entity description

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

ENTITY mux IS
    PORT (a, b, sel : IN std_logic;
        y : OUT std_logic);
END mux;
```
Architecture

- The ARCHITECTURE defines the function or behavior or structure of the ENTITY

- Consists of concurrent statements, e.g.
  - Process statements
  - Concurrent Signal Assignment statements
  - Conditional Signal Assignment statements

- An entity may have several architectures

```vhdl
ARCHITECTURE behavior OF mux IS
BEGIN
  y <= a WHEN sel = '0' ELSE b;
END behavior;
```
VHDL Notes

• There is no explicit reference to actual hardware components
  – There are no D-type flip-flops, mux, etc
  – Required logic is inferred from the VHDL description
  – Same VHDL can target many different devices

• There are many alternative ways to describe the required behavior of the final system
  – Exactly the same hardware will be produced
  – Some ways are more intuitive and easier to read

• Remember that the synthesis tools must be able to deduce your intent and system requirements
  – For sequential circuits it is usually necessary to follow recommended templates and style
Programmable Logic Devices

• Xilinx user programmable devices
  – FPGAs – Field Programmable Gate Array
    • Virtex 4, 5, 6, Kintex-7!
    • Spartan 3, Spartan 6, Artix-7
    • Consist of configurable logic blocks
      – Provides look-up tables to implement logic
      – Storage devices to implement flip-flops and latches
  – CPLDs – Complex Programmable Logic Devices
    • CoolRunner-II CPLDS (1.8 and 3.3 volt devices)
    • XC9500 Series (3.3 and 5 volt devices)
    • Consist of macrocells that contain programmable and-or matrix with flip-flops
• Altera has a similar range of devices
Electronic Components (Xilinx)

Acronyms
SPLD = Simple Prog. Logic Device
PAL = Prog. Array of Logic
CPLD = Complex PLD
FPGA = Field Prog. Gate Array

Common Resources
Configurable Logic Blocks (CLB)
- Memory Look-Up Table
- AND-OR planes
- Simple gates
Input / Output Blocks (IOB)
- Bidirectional, latches, inverters, pullup/pulldowns
Interconnect or Routing
- Local, internal feedback, and global
# Xilinx Products (Xilinx)

## CPLDs and FPGAs

<table>
<thead>
<tr>
<th>Complex Programmable Logic Device (CPLD)</th>
<th>Field-Programmable Gate Array (FPGA)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>PAL/22V10-like Gate array-like</td>
</tr>
<tr>
<td>More Combinational</td>
<td>More Registers + RAM</td>
</tr>
<tr>
<td><strong>Density</strong></td>
<td>Low-to-medium 0.5-10K logic gates</td>
</tr>
<tr>
<td></td>
<td>Medium-to-high 1K to 3.2M system gates</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>Predictable timing Up to 250 MHz today</td>
</tr>
<tr>
<td></td>
<td>Application dependent Up to 200 MHz today</td>
</tr>
<tr>
<td><strong>Interconnect</strong></td>
<td>“Crossbar Switch”</td>
</tr>
<tr>
<td></td>
<td>Incremental</td>
</tr>
</tbody>
</table>
Overview of Xilinx FPGA Architecture (Xilinx)
Spartan-6/Artix-7 FPGA Family

- “Specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications.
- “Delivers an optimal balance of low risk, low cost, and low power for cost-sensitive applications”
- Offers advanced power management technology, up to 150K logic cells, PCI express blocks, memory support, DSP slices, and transceivers
- DSP48A slice contains 18 by 18 multiplier, adder and accumulator

<table>
<thead>
<tr>
<th>Device</th>
<th>CLBs slices</th>
<th>CLB flip-flops</th>
<th>DSP48A slices</th>
<th>Block Ram (18 Kb)</th>
<th>User IO</th>
<th>Price (1 off)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC6LX16</td>
<td>2,278</td>
<td>18,224</td>
<td>32</td>
<td>32</td>
<td>232</td>
<td>$30</td>
</tr>
<tr>
<td>XC7A35T</td>
<td>5,200</td>
<td>41,600</td>
<td>90</td>
<td>100</td>
<td>106</td>
<td>$39</td>
</tr>
<tr>
<td>XC7A100T</td>
<td>15,850</td>
<td>126,800</td>
<td>240</td>
<td>135</td>
<td>210</td>
<td>$123</td>
</tr>
</tbody>
</table>
Programmable Functional Elements

- Configurable Logic Blocks (CLBs)
  - RAM-based look-up tables to implement logic
  - Storage elements for flip-flops or latches
- Input/Output Blocks
  - Supports bidirectional data flow and 3-state operation
  - Supports different signal standards including LVDS
  - Double-data rate registers included
  - Digitally controlled impedance provides on-chip terminations
- Block RAM provides data storage
  - 18-Kbit dual-port blocks
- Multiplier blocks (accepts two 18-bit binary numbers)
- Digital Clock Manager (DCM)
  - Provides distribution, delaying, mult, div, phase shift of clocks
Slices and CLBs (Xilinx)

• Each Virtex™-II CLB contains four slices
  – Local routing provides feedback between slices in the same CLB, and it provides routing to neighboring CLBs
  – A switch matrix provides access to general routing resources
Simplified Slice Structure (Xilinx)

- Each slice has four outputs
  - Two registered outputs, two non-registered outputs
  - Two BUFTs associated with each CLB, accessible by all 16 CLB outputs
- Carry logic runs vertically, up only
  - Two independent carry chains per CLB
Detailed Slice Structure (Xilinx)

- The next slides will discuss the slice features
  - LUTs
  - MUXF5, MUXF6, MUXF7, MUXF8 (only the F5 and F6 MUX are shown in the diagram)
  - Carry Logic
  - MULT_ANDs
  - Sequential Elements
Look-Up Tables (Xilinx)

- Combinatorial logic is stored in Look-Up Tables (LUTs)
  - Also called Function Generators (FGs)
  - Capacity is limited by number of inputs,
    - not complexity
- Delay through the LUT is constant
Flexible Sequential Elements (Xilinx)

- Can be flip-flops or latches
- Two in each slice; eight in each CLB
- Inputs can come from LUTs or from an independent CLB input
- Separate set and reset controls
  - Can be synchronous or asynchronous
- All controls are shared within a slice
  - Control signals can be inverted locally within a slice
IOB Element (Xilinx)

- **Input path**
  - Two DDR registers

- **Output path**
  - Two DDR registers
  - Two 3-state enable DDR registers

- **Separate clocks and clock enables for I and O**

- **Set and reset signals are shared**
SelectIO Standard (Xilinx)

- Allows direct connections to external signals of varied voltages and thresholds
  - Optimizes the speed/noise tradeoff
  - Saves having to place interface components onto your board
- Differential signaling standards
  - LVDS, BLVDS, ULVDS
  - LDT
  - LVPECL
- Single-ended I/O standards
  - LVTTL, LVCMOS (3.3V, 2.5V, 1.8V, and 1.5V)
  - PCI-X at 133 MHz, PCI (3.3V at 33 MHz and 66 MHz)
  - GTL, GTLP
  - and more!
Digital Controlled Impedance (DCI)

• DCI provides
  – Output drivers that match the impedance of the traces
  – On-chip termination for receivers and transmitters

• DCI advantages
  – Improves signal integrity by eliminating stub reflections
  – Reduces board routing complexity and component count by eliminating external resistors
  – Internal feedback circuit eliminates the effects of temperature, voltage, and process variations
Block SelectRAM Resources (Xilinx)

- Up to 3.5 Mb of RAM in 18-kb blocks
  - Synchronous read and write
- True dual-port memory
  - Each port has synchronous read and write capability
  - Different clocks for each port
- Supports initial values
- Synchronous reset on output latches
- Supports parity bits
  - One parity bit per eight data bits
Dedicated Multiplier Blocks (Xilinx)

- 18-bit twos complement signed operation
- Optimized to implement multiply and accumulate functions
- Multipliers are physically located next to block SelectRAM™ memory

![Diagram of 18x18 Multiplier with data inputs and output]

<table>
<thead>
<tr>
<th>Data_A (18 bits)</th>
<th>Data_B (18 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18 x 18 Multiplier</td>
</tr>
<tr>
<td></td>
<td>Output (36 bits)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 x 4 signed</td>
</tr>
<tr>
<td>8 x 8 signed</td>
</tr>
<tr>
<td>12 x 12 signed</td>
</tr>
<tr>
<td>18 x 18 signed</td>
</tr>
</tbody>
</table>
Nexys3 Board ($139 – now $189)

Note: The Nexys3 is no longer in production and is not recommended for new installations. When the current stock is depleted, it will be discontinued.
Nexys4 Board ($159)

- Artix-7 FPGA 100T (XC7A100T-1CSG324C)
  - 15,850 logic slices (4-input LUTs and 8 flip-flops)
  - Block RAM
  - Six clock management tiles
  - 240 DSP slices
  - Internal speeds > 450MHz
  - On chip ADC (XADC)
**Xilinx Design Process (Xilinx)**

- **Step 1:** Design
  - Two design entry methods: HDL (Verilog or VHDL) or schematic drawings
- **Step 2:** Synthesize to create Netlist
  - Translates V, VHD, SCH files into an industry standard format EDIF file
- **Step 3:** Implement design (netlist)
  - Translate, Map, Place & Route
- **Step 4:** Configure FPGA
  - Download BIT file into FPGA
Xilinx Design Flow (Xilinx)

Plan & Budget → Create Code/Schematic → HDL RTL Simulation

Implement

Translate → Map → Place & Route

Attain Timing Closure → Functional Simulation

Synthesize to create netlist → Timing Simulation

Create Bit File
Program the FPGA (Xilinx)

• There are three ways to program an FPGA
  – Through a PROM device
    • You will need to generate a file that the PROM programmer will understand
  – Directly from the computer
    • Use the iMPACT configuration tool
      • (need JTAG)
    – Use USB connector
      • Digilent Adept tool
Decoder Tutorial Demo Example
VHDL Source Code

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity decoder is
  Port ( sel : in std_logic_vector(2 downto 0);
        y : out std_logic_vector(7 downto 0));
end decoder;

architecture Behavioral of decoder is
begin
  y <= "00000000" when sel = "000" else
       "00000010" when sel = "001" else
       "00000100" when sel = "010" else
       "00001000" when sel = "011" else
       "00010000" when sel = "100" else
       "00100000" when sel = "101" else
       "01000000" when sel = "110" else
       "10000000";
end Behavioral;
```

Synthesizing the Design

---

*HDL Synthesis*

Synthesizing Unit <decoder>.
Related source file is "C:/ee574/nexsys2/decoder/decoder.vhd".
Found 1-of-8 decoder for signal <y>.
Summary:
inferred 1 Decoder(s).
Unit <decoder> synthesized.
View the Schematic Representation
Decoder Implemented on FPGA
Zooming in on Logic Slice
Assigning Package Pins

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Pin Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID</td>
<td>User ID</td>
</tr>
<tr>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>VCCINT</td>
<td>VCCINT</td>
</tr>
<tr>
<td>VCCO</td>
<td>VCCO</td>
</tr>
<tr>
<td>CONFIG</td>
<td>CONFIG</td>
</tr>
<tr>
<td>JTAG</td>
<td>JTAG</td>
</tr>
<tr>
<td>GCLK / GCK</td>
<td>GCLK / GCK</td>
</tr>
<tr>
<td>Power Management</td>
<td>Power</td>
</tr>
<tr>
<td>Not Connected</td>
<td>Not</td>
</tr>
<tr>
<td>Bank0</td>
<td>Bank0</td>
</tr>
<tr>
<td>Bank1</td>
<td>Bank1</td>
</tr>
<tr>
<td>Bank2</td>
<td>Bank2</td>
</tr>
<tr>
<td>Bank3</td>
<td>Bank3</td>
</tr>
<tr>
<td>Bank4</td>
<td>Bank4</td>
</tr>
<tr>
<td>Bank5</td>
<td>Bank5</td>
</tr>
<tr>
<td>Bank6</td>
<td>Bank6</td>
</tr>
<tr>
<td>Bank7</td>
<td>Bank7</td>
</tr>
</tbody>
</table>

In the diagram, the package pins are color-coded to indicate their function. The layout shows the top view of the package with pin numbers and labels for easy identification.
New Implementation to Match Target
Verilog Background

• 1983: Gateway Design Automation released Verilog HDL “Verilog” and simulator
• 1985: Verilog enhanced version – “Verilog-XL”
• 1987: Verilog-XL becoming more popular (same year VHDL released as IEEE standard)
• 1989: Cadence bought Gateway
• 1995: Verilog adopted by IEEE as standard 1364
  – Verilog HDL, Verilog 1995
• 2001: First major revision (cleanup and enhancements)
  – Standard 1364-2001 (or Verilog 2001)
• System Verilog under development
Create Verilog Module
Module Created

- No separate entity and arch – just module
- Ports can be input, output, or inout
- Note: Verilog 2001 has alternative port style:
  - (input a, b, sel, output y);
Add Assign Statement

- Similar to VHDL conditional signal assignment – continuous assignment
- Exactly same hardware produced
Verilog - General Comments

- VHDL is like ADA and Pascal in style
  - Strongly typed – more robust
- Verilog is more like the ‘C’ language
- Verilog is case sensitive
- White space is OK (tabs, new lines, etc)
- Statements terminated with semicolon (;)
- Verilog statements between
  - module and endmodule
- Comments // single line and /* and */
Verilog Logic

- Four-value logic system
  - 0 – logic zero, or false condition
  - 1 – logic 1, or true condition
  - x, X – unknown logic value
  - z, Z - high-impedance state

- Number formats
  - b, B binary
  - d, D decimal (default)
  - h, H hexadecimal
  - o, O octal

  - 16’H789A – 16-bit number in hex format
  - 1’b0 – 1-bit
Verilog and VHDL – Reminder

- VHDL - like Pascal and Ada programming languages
- Verilog - more like ‘C’ programming language
- But remember they are Hardware Description Languages - They are NOT programming languages
  - FPGAs do NOT contain an hidden microprocessor or interpreter or memory that executes the VHDL or Verilog code
  - Synthesis tools prepare a hardware design that is inferred from the behavior described by the HDL
  - A bit stream is transferred to the programmable device to configure the device
  - No shortcuts! Need to understand combinational/sequential logic
- Uses subset of language for synthesis
- Check - could you design circuit from description?