Verilog – Module 1

Introduction

Jim Duckworth
ECE Department, WPI
Topics

- Background to Verilog
- Introduction to language
- Programmable Logic Devices
  - CPLDs and FPGAs
  - FPGA architecture
  - Nexys 3 Board (with Spartan 6 FPGA) [discontinued]
  - Basys 3 Board (with Artix-7 FPGA)
  - Nexys4DDR Board (with Artix-7 FPGA)
- Using Verilog 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.
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
  - Better system simulation and verification support
Books

- “HDL Chip Design” by Smith, 1996, Doone Publications, 0-9651934-8
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:
  - \( \text{module mux(a,b,sel,y);} \)
    - \( (\text{input a, b, sel, output y);} \)
      - Also place in column:
        - (input a, b, sel, output y);
        - (input a, b, sel, output y);
Add assign statement

- Similar to VHDL conditional signal assignment – continuous assignment
- Same hardware produced as with VHDL
Verilog - general comments

- VHDL is like ADA and Pascal in style
  - Strongly typed – more robust than Verilog
  - In Verilog it is easier to make mistakes
    - Watch for signals of different widths
    - No default required for case statement, etc
- Verilog is more like the ‘c’ language
- Verilog IS case sensitive
- White space is OK
- Statements terminated with semicolon (;)
- Verilog statements between
  - module and endmodule
- Comments // single line and /* and */
Verilog

- 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
Example Synthesis Results (not Xilinx)
Verilog 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

**Source:** Dataquest
# 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>Architecture</td>
<td>Architecture</td>
</tr>
<tr>
<td>PAL/22V10-like</td>
<td>Gate array-like</td>
</tr>
<tr>
<td>More Combinational</td>
<td>More Registers + RAM</td>
</tr>
<tr>
<td>Density</td>
<td>Density</td>
</tr>
<tr>
<td>Low-to-medium</td>
<td>Medium-to-high</td>
</tr>
<tr>
<td>0.5-10K logic gates</td>
<td>1K to 3.2M system gates</td>
</tr>
<tr>
<td>Performance</td>
<td>Performance</td>
</tr>
<tr>
<td>Predictable timing</td>
<td>Application dependent</td>
</tr>
<tr>
<td>Up to 250 MHz today</td>
<td>Up to 200 MHz today</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Interconnect</td>
</tr>
<tr>
<td>“Crossbar Switch”</td>
<td>Incremental</td>
</tr>
</tbody>
</table>

Jim Duckworth, WPI  
Module 1
Overview of Xilinx FPGA Architecture (Xilinx)

IOB
Input Output Block

CLB
Configurable Logic Block

PSM
Programmable Switch Matrix

Connection lines
Single, Long
Double, Direct
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)

Figure 3: Diagram of SLICEM
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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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 18 x 18 Multiplier with inputs and outputs](image-url)

<table>
<thead>
<tr>
<th>Multiplication 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.
Basys 3 Artix-7 Board ($79)

• Based on latest Artix-7 FPGA (XC7A35T-1CPG236C)
Nexys 4 DDR Board ($159)

- Based on latest Artix-7 FPGA (XC7A100T-1CSG324C)
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)
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
Verilog Source Code

```verilog
// Additional Comments:

module decoder(
    input [2:0] sw,
    output [7:0] led
);

assign led = (sw == 3'b000) ? 8'b00000001 :
            (sw == 3'b001) ? 8'b00000010 :
            (sw == 3'b010) ? 8'b00000100 :
            (sw == 3'b011) ? 8'b00001000 :
            (sw == 3'b100) ? 8'b00010000 :
            (sw == 3'b101) ? 8'b00100000 :
            (sw == 3'b110) ? 8'b01000000 :
            8'b10000000;

endmodule
```
Synthesizing the Design

Synthesizing Unit <decoder>.
Related source file is "C:\ece3829\decoder\decoder.v".
Found 8x8-bit Read Only RAM for signal <led>

Summary:
  inferred 1 RAM(s).
Unit <decoder> synthesized.
View the Schematic Representation
Decoder Implemented on FPGA
Zooming in on Logic Slice

For Help, press F1
Assigning Package Pins
New Implementation to Match Target
Top-Down Design Hierarchy

- Instantiate module (counter example with decoder)

```verilog
module decoder(
    input [3:0] count,
    output [6:0] seven_seg
);
```

// instantiate decoder module in counter
// using position of ports (positional association)
decoder d1 (count_val, seven_seg_val);

// or using formal and actual names (named association)
decoder d1 (.count(count_val), .seven_seg(seven_seg_val));
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?