Introduction to Verilog

Dov Kruger

Department of Electrical and Computer Engineering
Rutgers University

September 24, 2024

What is Verilog?

- Hardware Description Language (HDL)
- Used to model digital systems
- Part of the C family of languages
- VHDL is a competitor, based on Ada
- Widely used for FPGA and ASIC design
- Supports behavioral, structural, and gate-level modeling

Versions of Verilog

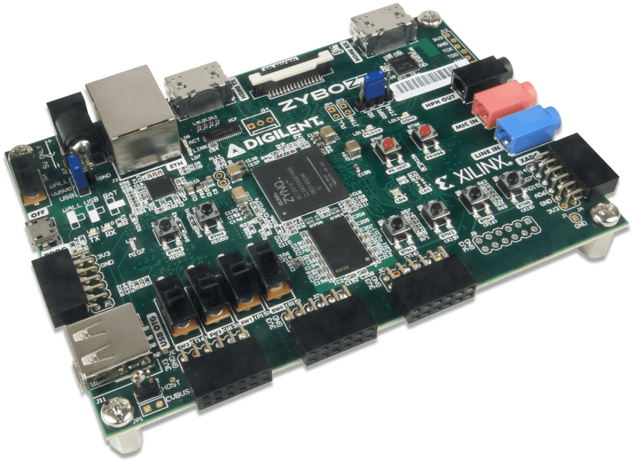
- 1984: Verilog introduced by Gateway Design Automation
- 1995: IEEE standardized Verilog as IEEE 1364-1995
- 2001: Verilog-2001, major enhancements including generate blocks
- 2005: Verilog-2005, minor updates and clarifications
- 2009: SystemVerilog, extended to include object-oriented features
- 2012: IEEE 1800-2012, latest version with enhanced features for verification and design

What is an FPGA?

- Field-Programmable Gate Array (FPGA)
- Configurable hardware components: logic blocks, I/O blocks, routing
- Used for prototyping, custom hardware, and flexible digital designs
- Can be reprogrammed after manufacturing
- Commonly used in embedded systems, signal processing, and more
- Much higher performance per watt than a computer

Example Board

Z7000 FPGA Board, $399 (2024)



FPGA Models and Specifications

Model	Year	Clock	Units	RAM	Price	Power
Xilinx Zynq 7000	2018	667 MHz	23k	1GB DDR3	$399	1.5W
Xilinx UltraScale+	2017	1.2 GHz	2.5M	5GB DDR4	$3962	62W

Verilog Usage in This Course

- We will focus on writing Verilog code
- Not programming actual FPGAs in the lab
- Verilog allows us to design complex circuits beyond basic lab setups
- FPGA access available upon request for additional hands-on practice

FPGA vs ASIC

- Application Specific Integrated Circuit (ASIC) are custom and must be built to order
- Requires volume, lead time, and high upfront cost
- FPGA are configurable after manufacturing
- FPGA are less power efficient than ASICs
- FPGA are larger than ASICs
- FPGA are more expensive than ASICs

Verilog Compilers/Simulators

- Several options are available, each with its own advantages and drawbacks.
- We will use Icarus Verilog, an open-source Verilog compiler and simulator.

Name	Type	Download Site	Notes
Icarus Verilog	Open Source	`http://iverilog.icarus.com`	Lightweight, easy to use
Xilinx Vivado	Commercial	`https://www.xilinx.com`	Industry standard, large install
ModelSim	Commercial	`https://www.mentor.com`	Widely used, powerful features

Additional Information on Verilog Tools

- **Icarus Verilog:** Best for small projects and learning environments. Limited support for advanced SystemVerilog features.
- **Xilinx Vivado:** Essential for FPGA development, especially with Xilinx hardware. Includes extensive support for IP cores and other hardware features.
- **ModelSim:** Excellent debugging and waveform analysis tools. Can handle large and complex designs.
- **Verilator:** Not a traditional simulator; it translates Verilog into C++/SystemC for fast simulation. Best for projects needing high-speed simulation and integration with other C++ systems.
- **Synopsys VCS:** Premium tool with advanced verification capabilities, including formal verification and coverage analysis. Used in professional ASIC/FPGA design.

Naming Conventions in Verilog

- `.v` — Verilog source files
- `.sv` — SystemVerilog source files
- `_tb.sv` — Testbench files (used for simulation)
- Names should be descriptive of the module's function
- Testbench files simulate how the circuit behaves (similar to a "main" in programming)

First Program

```
`timescale 1ns/1ns

module first_example;
  reg a, b;
  wire result;
  assign result = a & b; // AND gate

  initial begin
    $monitor("Time = %0t : a = %b, b = %b, result =
    a = 0; b = 0;
    #10 a = 0; b = 1;
    #10 a = 1; b = 0;
    #10 a = 1; b = 1;
    #10 $finish;
  end
endmodule
```

Data Types Overview

The following code shows integer data types and how to print to the screen

```
module datatypes_overview_tb;
  reg [7:0] a;
  reg [15:0] b;
  logic signed [15:0] c;
  logic signed [31:0] d;
  logic signed [63:0] e;

 initial begin
  a = 8'hAA;
  b = 16'h5555;
  c = 16'hFFFF;
  d = 123;
  e = 64'h1234567812345678;

  $display("byte (8-bit):          %b", a);
```

Bitwise Operations

The following example shows bitwise operations

```
module bitwise_ops_tb;
    reg [3:0] a = 4'b1010;
    reg [3:0] b = 4'b1100;

    initial begin
        $display("a & b = %b", a & b);
        $display("a | b = %b", a | b);
        $display("a ^ b = %b", a ^ b);
        $display("~a    = %b", ~a);
        $finish;
    end
endmodule
```

NAND Gate

```
module nand_gate(
    input wire a,
    input wire b,
    output wire y
);
    assign y = ~(a & b);
endmodule
```

NAND Gate TestBench

```
module nand_tb;
  reg a, b;
  wire y;
  nand_gate uut (  // instantiate (create) the NAND
    .a(a),
    .b(b),
    .y(y)
  );
  initial begin
    a = 0; b = 0; #1 $display("NAND(0,0) = %b", y);
    a = 0; b = 1; #1 $display("NAND(0,1) = %b", y);
    a = 1; b = 0; #1 $display("NAND(1,0) = %b", y);
    a = 1; b = 1; #1 $display("NAND(1,1) = %b", y);
    $finish;
  end
endmodule
```

NAND Gate, v2

assignment without keyword "assign" must be inside an "always" block

```
module nand_gate(
    input wire a,
    input wire b,
    output wire y
);
  always @(*) begin
    y = ~(a & b);
  end
endmodule
```

Example: Majority Vote

- Write a module that has 3 inputs A,B,C
- returns 1 if 2 of the 3 inputs are 1, 0 otherwise
- How will you test this module?