Verilog Introduction

Prof. Jakub Szefer
Dept. of Electrical Engineering, Yale University

EENG 428 / ENAS 968
Cloud FPGA
Verilog – A Tutorial Introduction

This lecture is mostly based on contents of Chapter 1, from “The Verilog Hardware Description Language” book [1], 5th edition. Example figures and (modified) code are from the textbook unless otherwise specified.

Textbook topics covered:
• Structural description of circuits
• Combinatorial circuit description
• Sequential circuit description
• Module hierarchy
• Seven segment module and testbench examples

Additional topics:
• Very short history of Verilog
• Very short intro to iverilog and gtkwave open-source tools
Verilog is a Hardware Description Language (HDL) used to describe and model electronic systems built using digital logic

- Can be used to do some analog and mixed-signal designs
- Originally developed only for description and simulation of circuits
- Latter features were added to allow synthesis of the design into hardware

**Verilog standards:**
- IEEE Standard 1364-1995 – initial Verilog
- IEEE Standard 1364-2001 – standard used in the textbook
- IEEE Standard 1364-2005 – minor updates

**SystemVerilog standards:**
- IEEE Standard 1800-2009 – merge SystemVerilog and Verilog into one standard
- IEEE Standard 1800-2012 – minor updates
  - Verilog is now subset of SystemVerilog
(Ch. 1) Verilog – A Tutorial Introduction
Description of a Module

A module is a basic unit of hardware, which performs some combinatorial or sequential logic.

Most often we use block diagrams to describe modules:

- **Inputs** (typically on left side of figure):
  - A
  - B
  - C
  - D
  - E
  - F

- **Outputs** (typically on right side of figure):
  - eSeg

- **ModuleName**

  - **clk**
  - **rst_n**

- **Notes**:
  - Should indicate bit width if the signal is more than 1 bit.
  - Most sequential logic modules have clock (clk) and reset (rst, or rst_n if active low).
  - The clk and rst are often not shown explicitly on block diagrams.
  - Combinatorial-only logic modules do not need clock or reset.
**Structural Description of a Module**

In Verilog modules are described using the `module` keyword and include description of inputs and outputs (not shown in this example), wires, registers, sub-modules, and any combinatorial or sequential logic.

```verilog
module binaryToESeg
    ( input A, B, C, D,
      output eSeg
    );

    wire p1, p2, p3, p4;

    nand #1
        g1 (p1, C, ~D),
        g2 (p2, A, B),
        g3 (p3, ~B, ~D),
        g4 (p4, A, C),
        g5 (eSeg, p1, p2, p3, p4);

endmodule
```

Specify delay of this gate (for simulation only)
Adding Ports to a Module

All Verilog modules, except a top-level testbench module will have inputs and outputs:
- Inputs are wires coming into module
- Outputs can be wires or registers which go out of module
- Port names are only in-scope inside the module

In Verilog-2001 ports and their type are defined at beginning of module, no need to define ports and latter define their type and size (older Verilog style)

Example:
```
module binaryToESeg
    (input A, B, C, D,
     output eSeg
    );

    wire p1, p2, p3, p4;

    nand #1
        g1 (p1, C, ~D),
        g2 (p2, A, B),
        g3 (p3, ~B, ~D),
        g4 (p4, A, C),
        g5 (eSeg, p1, p2, p3, p4);

endmodule
```
Verilog Built-In Primitive Gates and Operators

Operators:
- Verilog-2001 standard supports many logical and arithmetic operators
  - E.g. unary operators: +, -, !, ~, &, ~&; |, ~|, ^, ~^, ^^
  - E.g. binary operators: +, -, *, /, %, ==, !=, ===, !==, and more

Primitive gates:
- Verilog-2001 standard supports a number of primitive gates (these do not need to be built from logic operators):
  - E.g. n-input gates: and, nand, or, nor, xor, xnor

See Appendix G of the textbook for full formal specification of the language, built-in gates, and operators.

Structural description is built using the primitive gates, operators, and modules or submodules made of these.
IP (Intellectual Property) cores:
- Most tools and vendors provide IP cores which realize many common types of operations
  - PCIe controller, DRAM controller, AXI bus controller, FIFOs, memories, etc.
  - Depend on the tools, vendor, and may be specific to one FPGA chip or board type
  - Can be used to more quickly build a design
  - Can create your own IP core
Simulating a Module and Testbench Design
Simulating a Module

Operation of a module can be examined by creating a simulation that:
- Provides stimuli to the module being tested
- Can capture the outputs (or any changes in the internal state)

Simulation using a testbench module:
- Can create a Verilog module (testbench) that provides stimuli to the design under test
- Use `initial` statements to assign values to registers
- Use `$monitor` statement to print outputs (values of registers or wires)

```verilog
module binaryToESegSim;

// design under test (DUT): seven segment
// display driver for segment e
wire eSeg, p1, p2, p3, p4;
reg A, B, C, D;

nand #1
  g1 (p1, C, ~D),
  g2 (p2, A, B),
  g3 (p3, ~B, ~D),
  g4 (p4, A, C),
  g5 (eSeg, p1, p2, p3, p4);

initial
begin
  $monitor ($time,,
  "A = %b B = %b C = %b D = %b, eSeg = %b",
  A, B, C, D, eSeg);

// simulate input changes to drive the DUT
#10 A = 0; B = 0; C = 0; D = 0;
#10 D = 1;
#10 C = 1; D = 0;
#10 $finish;
end
endmodule
```

DUT (Design Under Test) would typically be an instance of a module you want to test.
Simulation will typically have two types of output:

- Text printed to terminal
- A waveform which shows how the digital signals change in time
  - Actually a waveform file, which can be viewed using a waveform viewer program

**Terminal output:**
- Use `$monitor` statements to generate output

**Waveform output:**
- Use `$dumpfile` statements to write signal values to dump file
Simulation Output and Waveforms

Testbench output is simply statements printed to the terminal
- Example output for the seven segment decoder

```
0  A = x  B = x  C = x  D = x,  eSeg = x
10 A = 0  B = 0  C = 0  D = 0,  eSeg = x
12 A = 0  B = 0  C = 0  D = 0,  eSeg = 1
20 A = 0  B = 0  C = 0  D = 1,  eSeg = 1
22 A = 0  B = 0  C = 0  D = 1,  eSeg = 0
30 A = 0  B = 0  C = 1  D = 0,  eSeg = 0
32 A = 0  B = 0  C = 1  D = 0,  eSeg = 1
```

Waveforms are the time-changing values of the signals
- Example waveform for the seven segment decoder
Creating a Testbench for a Module

Simulation of a module (or multiple modules) can be organized using a testbench
• Testbench is a top-level module
• Drive inputs to design under test (DUT) submodule or submodules
• Capture output of the submodule
  • Can also monitor internal submodule state

Testbench approach separates the testing from the module being tested
• Goal is to test module that will be synthesized into hardware (FPGA or ASIC)
• Don’t modify design under test module
• Only give inputs and observer the outputs
Creating a Testbench for a Module

Example testbench for seven segment module

- Top-level module is the testbench
- It has no inputs or outputs
- Connects DUT to the tester
- Tester module also outputs print statements to terminal

```
module testBench;
    wire w1, w2, w3, w4, w5;
    binaryToESeg d (w1, w2, w3, w4, w5);
    test_bToESeg t (w1, w2, w3, w4, w5);
endmodule
```

```
// design under test (DUT): seven segment
// display driver for segment e
module binaryToESeg
    ( input A, B, C, D, 
    output eSeg 
    );
    wire p1, p2, p3, p4;
    
    // #1
    
    // #2
    
    // #3
    
    // #4
    
    // #5
    
    endmodule
```

```
// tester unit: drive inputs to DUT
module test_bToESeg
    ( output reg A, B, C, D, 
    input eSeg 
    );
    
    initial
    begin
    $monitor ($time,
    "A = %b B = %b C = %b D = %b, eSeg = %b",
    A, B, C, D, eSeg);
    
    // simulate input changes to drive
    // the design under test
    #10 A = 0; B = 0; C = 0; D = 0;
    #10 B = 1;
    #10 C = 1; D = 0;
    #10 $finish;
    end
endmodule
```
Modeling of Digital Circuits
Subsets of Verilog

Verilog can be used for both synthesis and simulation

- Write description which is used to generate hardware (FPGA or ASIC); use the synthesizable subset of Verilog
- Write description which is used to simulate behavior of the modules, simulate timing (sometimes power); use behavioral modeling

Share: bit.ly/cloudfpga
In a combinatorial circuit there is no “memory” as output depends on
• Output only depends on current inputs
• No “memory” in the circuit

Rules for combinatorial circuits code
• Make sure all inputs are in the sensitivity list @ (...)
  • Can use @ (*) as short hand
• Make sure all possible executions of the always block assign a value to the output

module binaryToEsSeg_Behavioral
(output reg eSeg,
input A, B, C, D);

always @(A, B, C, D) begin
  eSeg = 1;
  if (~A & D) eSeg = 0;
  if (~A & B & ~C) eSeg = 0;
  if (~B & ~C & D) eSeg = 0;
end
endmodule

eSeg is defined as reg because it is used on the left-hand-side in always block; but it is still combinatorial logic

Behavioral description can be synthesized to hardware, usually
Behavioral Modeling of Sequential Circuits

Sequential circuits have “memory” and output depends on:

• Current inputs
• And state of the circuit

• State is stored in flip-flops
  • Represented by `reg` in Verilog
  • Typically D flip-flops

Combinatorial logic, also called next-state logic

Flip-flops storing the state of the circuit

module fsm
(output reg out,
input in, clock, reset
);

reg [1:0] currentState, nextState;

// the combinational portion
always @(in, currentState)
begin
  out = ~currentState[1] & currentState[0];
  nextState = 0;
  if (currentState == 0)
    if (in) nextState = 1;
  if (currentState == 1)
    if (in) nextState = 3;
  if (currentState == 3)
    begin
      if (in) nextState = 3;
      else nextState = 1;
    end
  end
end

// the sequential portion
always @(posedge clock, negedge reset)
begin
  if (~reset)
    currentState <= 0;
  else
    currentState <= nextState;
end
endmodule

---

Share:
bit.ly/cloudfpga

EENG 428 / ENAS 968 – Cloud FPGA
© Jakub Szefer, Fall 2022
Rules for sequential code for always blocks implementing the state registers

- Sensitivity list of `always` block for updating flip-flops only includes edges of clock, reset, and set signals

- In `always` block always specify reset and set conditions first
  - This will give you asynchronous reset or set, that does not depend on the clock
  - Clock generator module can itself use reset, so usually want to be independent of the clock

- All signals on the left-hand-side need to be registers

- Use non-blocking `<=` operator to perform all assignments in parallel
  - If blocking `=` is used, latches may be synthesized

- Break up the state machine into the combinatorial logic and sequential logic
  - Can have combinatorial logic in the `always` block, but it will be evaluated only on clock edge
Module Hierarchy

All hardware designs are done in hierarchical manner
• Develop modules for specific tasks
• Interconnect multiple modules together
• Can have nesting of modules

Advantages
• Easier to organize the design (and code)
• Can update one module without having to update whole design
  • Assuming interface inputs and outputs stay same
  • Often use standard interfaces such as AXI for compatibility with other designs
• Treat modules as black-boxes
  • E.g. IP cores purchased
• Have different versions of each module (for initial simulation vs. final design)
Module Hierarchy Example

```verilog
module boardWithConcatenation;
    wire clock, eSeg, w3, w2, w1, w0;
    wire [3:0] countVal;

    assign countVal = {w3, w2, w1, w0};

    // sub modules
    m16 counter (w3, w2, w1, w0), clock);
    m555 clockGen (clock);
    binaryToESeg disp (eSeg, w3, w2, w1, w0);

    initial
    begin
        $monitor (${time,,","count=%d, eSeg=%d", countVal, eSeg});
        #200 $finish;
    end
endmodule

module m16
    ( output reg [3:0] ctr = 1,
      input clock
    );

    // counter ctr is initialized to 1 when devined,
    // then aways count up on clock edge
    always @(posedge clock)
    begin
        ctr <= ctr + 1;
    end
endmodule

module m555
    ( output reg clock
    );

    // initialize clock at 5th cycle of simulator
    initial
    begin
        #5 clock = 1;
    end

    // every 50 cycles of simulation, toggle the clock
    always
    begin
        #50 clock = ~ clock;
    end
endmodule

module binaryToESeg
    ( output eSeg,
      input A, B, C, D
    );

    wire p1, p2, p3, p4;
    nand #1
    g1 (p1, C, ~D),
    g2 (p2, A, B),
    g3 (p3, ~B, ~D),
    g4 (p4, A, C),
    g5 (eSeg, p1, p2, p3, p4);
endmodule
```

Textbook example of display driver with clock and counter

--

EENG 428 / ENAS 968 – Cloud FPGA
© Jakub Szefer, Fall 2022
Open-Source Verilog Tools
Icarus Verilog

Icarus Verilog, also called *iverilog*, is a free and open-source compiler and simulator for Verilog designs.

**Typical use of Icarus Verilog:**

- Compile a design, typically a testbench, into an assembly-like language
- Run the assembly-like language using the `vpp` command
  - Print statements on terminal to see progress of simulation, signal values, etc.
  - Optionally output a waveform, which can be viewed by `gtkwave` or other waveform viewer
- Can also generate a netlist
  - There is null target as well, useful for syntax checking

**As of 2022, the latest version is 12, version 10 is still common, it supports:**

- Supports Verilog standards 1364-1995, 1364-2001, and 1364-2005
- Does not seem to support latest 1800-2017

GTKWave

GTKWave, also called *gtkwave*, is a waveform viewer

- Visualize time-changing digital signals
- Support different input file types, including VCD
  - Value Change Dump (VCD) is ASCII-based format for dump files generated by electronic design automaton (EDA) tools

**Typical use of GTKWave:**

- Generate a dumpfile using EDA tools
  - Usually part of testbench is to dump the waveform files
- Run the viewer to observe signal changes in time
- Use waveform to help debug the design

- Waveforms are useful for design-time debugging
- Reading raw dump files or raw output from testbench is more useful for automated testing

GTKWave logo from: http://gtkwave.sourceforge.net/
gEDA project is a set of electronic design automation tools released under the GPL open-source license

- Icarus Verilog - A Verilog simulator
- GTKWave - A digital waveform viewer
- ngspice - a port of Berkeley SPICE
- PCB - PCB layout program
- Etc.

Open Circuit Design project is another set of design automation tools, also free or open-source, but focusing more on VLSI and ASIC

- Magic - VLSI layout editor, extraction, and DRC tool
- Xcircuit - circuit drawing and schematic capture tool
- IRSIM - switch-level digital circuit simulator
- Etc.

This list is wholly incomplete, more tools can be found by searching the internet...
Verilog Introduction Summary

Verilog was developed as simulation tool and language, did not support synthesis at first
• Both tool and language developed together
• Later Verilog became standardized
• Now many tools support the various Verilog standards

Verilog is used to describe a digital computing system as a set of modules
• Inputs and outputs of the module
• Description of what the module does

Synthesizable subset of Verilog are keywords and syntax which can be synthesized into a netlist that can latter be mapped to a specific hardware (FPGA, ASIC, etc.)

Module description
• Behavioral – describe the behavior of the module, more like a program or a function
• Structural – describe actual logical gates, how they are connected, etc.
References


EENG 428 / ENAS 968 – Cloud FPGA
© Jakub Szef er, Fall 2022