

ECE 551

Digital Design And Synthesis

Lecture 1

Fall'17

Instructor Introduction
Course Introduction
Administrative Stuff
Introduction to Verilog

Instructors

Eric Hoffman

- Faculty Associate
- Have no PhD, Masters only
- 25+ years industry experience doing Integrated Circuit & System design
 - 10 years at Intel
 - 7+ years at ZMD (Mixed signal, Analog/Digital IC's)
 - 8+ years as independent consultant
- Instructing experience:
 - ECE555, ECE552, ECE551, ECE353

Dave Schlaiz = TA

2

Instructor and TA Office Hours

- Eric Hoffman
 - Office = EH2358, if not there then check 3654 or possibly B555
 - Hours
 - Mon & Weds 2:30 to 4:00 (also around other times. If door all the way open feel free)
 - Email = erichoffman@wisc.edu
- TA = Dave Schlaiz
 - Office Hours = Held in B555 (basement unix lab)
 - Tuesdays 1:30 – 3:00
 - Available via email
 - Email = dschlaiz2@wisc.edu
- Discussion Sessions hosted by Dave
 - Monday (4:00 – 5:00) (TA has conflict with original time)(Room 2535)
 - Tuesday 6:30 – 7:30PM, Room is 2255
 - Some weeks discussion will be tutorials, in which case room is B555

3

Course Goals

- Provide knowledge and experience in:
 - Digital circuit design using a HDL (Verilog & SVerilog)
 - HDL simulation
 - Good practices in digital design verification
 - How to build self checking test benches
 - How to evaluate test suite using coverage tools
 - Synthesis of dataflow and behavioral designs
 - Optimizing hardware designs (timing, area, power)
 - Basic static timing analysis concepts
 - Design tools commonly used in industry
- Teach you to be able to “think hardware” first, then code it in an HDL.

4

Course Structure (not inverted, not standard)

- Some Video Lectures (with subsequent quizzes)
- Some Classroom Lectures (usually one a week)
- Class room exercises with: **ModelSim/Quartus/Synopsys**
- In class exercises
 - Most individual, some as team of 2, some as project team
 - Verilog DUT and testbench creation & simulation
 - Sometimes mapping it to Altera FPGA (DE0 Nano)
 - Sometime mapping it to std cell (Synopsys)
- First 2 classes will be lectures because we have to have background before exercises practical.

5

What You Should Already Know

- Principles of basic digital logic design (ECE 352)
 - Number representations (unsigned, signed, Hex & Binary)
 - Boolean algebra
 - Gate-level design
 - K-Map minimization
 - Finite State Machines
 - Basic datapath structures (adders, shifters, SRAM)
- How to log in to CAE machines and use a Linux shell

6

Course Website

- Moodle course page: (ay17-18.moodle.wisc.edu)
- What the Website will have:
 - Lecture Notes (I will try to stay 1 week ahead of class)
 - Lecture Videos and Quizzes
 - Homework Assignments
 - In Class Exercise Descriptions
 - Tutorials
 - Project Information
 - Supplemental Information
 - Midterm Solution

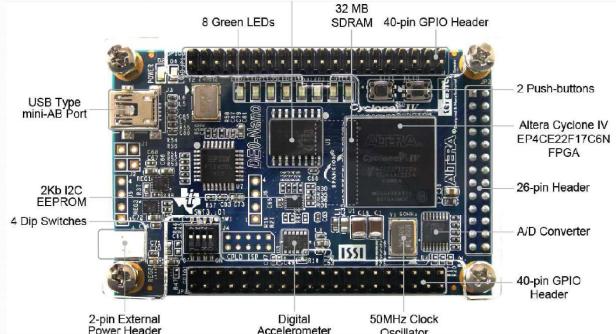
7

Course Materials

- Lecture Slides
- No Textbook Necessary
 - Can get by with the Standards & Lecture slides
- Standards
 - IEEE Std.1364-2001, IEEE Standard Verilog Hardware Description Language, IEEE, Inc., 2001.
 - IEEE Std 1364.1-2002, IEEE Standard for Verilog Register Transfer Level Synthesis, IEEE, Inc., 2002
- Synopsys on-line documentation
- Other useful readings
- Will have to buy a DE0 Nano FPGA board.
 - Pair up with someone and buy one for group of two.
 - Will sell them 3 weeks or so into the semester for \$70 each.

8

DE0-Nano (Altera FPGA Board)



Evaluation and Grading

- Approximately:

Graded on a curve. Class average grade will be around 3.25

- 12.5% Homework
 - 8% Quizzes on video lectures (lowest 1 score dropped)
 - 8% In class exercise results (lowest 2 scores dropped)
 - 1.5% Cummings paper quiz
 - 30% Project (groups of 3 or 4) (Establish team soon)
 - 20% Midterm
 - 20% Final
- Homework due before of class period of due date
- **15% penalty** for each late period of 24 hours
 - Not accepted >48 hours after deadline
 - Will be done as Moodle Quizzes or dropboxes.

10

Class Project

- Work in groups of 3 or 4 students
- Design, model, simulate, synthesize, and test a complex digital system.
- Several milestones
 - Forming teams
 - Homework problems that build toward the project
 - Project status report
 - Final demonstration (including mapping to your DE0 Nano)
- More details coming later in the course

11

Course Tools

- Industry-standard design tools:

- Modelsim HDL Simulation Tools (Mentor)
- Design Vision Synthesis Tools (Synopsys)
- TSMC 45nm CMOS Standard Cell Technology Library

■ Tutorials will be available for both tools

- Modelsim tutorials next week, (*only attend one of these*)
 - Monday 4:00 – 5:30 held in B555
 - Tuesday 6:30 – 8:00 held in B555
 - Wednesday 6:30 – 8:00 held in B555
- Design Vision tutorial a 6 to 7 weeks later
- Tool knowledge will be required to complete homeworks
- Dave Schlaits will be a resource for help on tools

12

ModelSim Tutorial

- ModelSim is our Verilog simulator.
 - Tutorial takes about 1 to 1.5 hours
 - 3 tutorial sessions will be held next week (*only attend one*)
 - Monday Sept 11th 4:00 – 5:30 in B555
 - Tuesday, Sept 12th 6:30 – 8:00 in B555
 - Wednesday, Sept 13th 6:30 – 8:00 in B555
 - You only need to attend one of these sessions!
 - You can also do it on your own if you can't make any of these sessions.

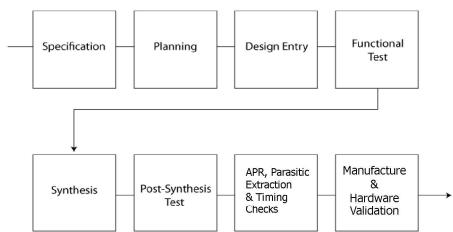
13

What You Should Get From This Class

- This class will teach you how to use Hardware Description Languages (HDLs) to design, verify, and realize digital logic
- You will learn how to synthesize HDLs into hardware using the same tools used in industry
- You will participate in the always enlightening process of working to design a digital system in a team environment
- By the end of this course, most of you should be qualified for an entry-level job or internship at a hardware design firm (\$\$\$)

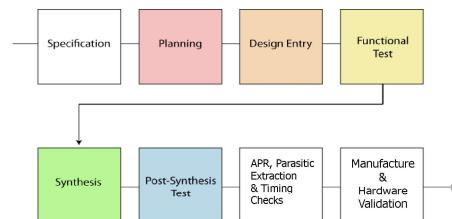
14

Digital Design Process



15

Description Phase



Implementation Phase

What is an HDL?

- HDL = Hardware Description Language
 - Allows for modeling & simulation (with timing) of digital designs
 - Can be synthesized into hardware (netlist) by synthesis tools (Synopsys, Ambit, FPGA compilers)
 - Two major standards in industry & academia
 - ✓ Verilog (Flexible, loose, more common in industry)
 - ✓ VHDL (Strongly typed, more common in defense and automotive)
 - ✓ Having used both I prefer Verilog. This course will use Verilog
 - ✓ System Verilog (Sverilog) is the next generation of verilog, and we will be learning portions of system verilog as well.

17

What is an HDL? (continued)

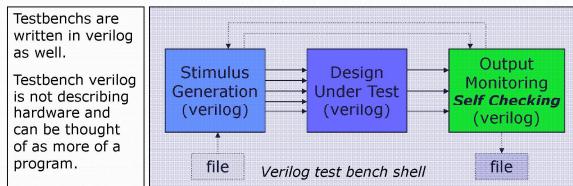
```
module counter(clk,rst_n,cnt);
  input clk,rst_n;
  output [3:0] cnt;
  reg [3:0] cnt;
  always @(posedge clk) begin
    if (!rst_n)
      cnt = 4'b0000;
    else
      cnt = cnt+1;
  end
endmodule
```

- It looks like a programming language
- It is **NOT** a programming language
 - ✓ It is always critical to recall you are describing hardware
 - ✓ This codes primary purpose is to generate hardware
 - ✓ The hardware this code describes (a counter) can be simulated on a computer. In this secondary use of the language it does act more like a programming language.

18

Simulating/Validating HDL

- The sad truth...
 - 10% design, 90% validation
 - If you do it right you will spend 9X more time testing/validating a design than designing it.

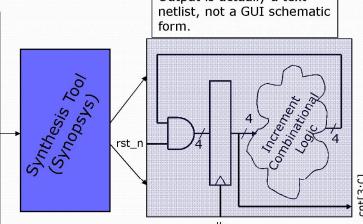


19

What is Synthesis?

- Takes a description of what a circuit DOES
- Creates the hardware to DO it

```
module counter(clk,rst_n,cnt);
  input clk,rst_n;
  output [3:0] cnt;
  reg [3:0] cnt;
  always @(posedge clk) begin
    if (!rst_n)
      cnt = 4'b0000;
    else
      cnt = cnt+1;
  end
endmodule
```

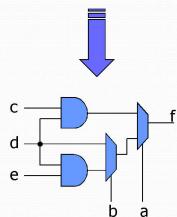


20

Synthesizing the Hardware Described

- All hardware created during synthesis
 - Even if a is true, still computing d&e
- Learn to understand how descriptions translated to hardware

```
if (a) f = c & d;  
else if (b) f = d;  
else f = d & e;
```



21

Why Use an HDL?

- Enables Larger Designs
 - More abstracted than schematics, allows larger designs.
 - Register Transfer Level Description
 - Wide datapaths (16, 32, or 64 bits wide) can be abstracted to a single vector
 - Synthesis tool does the bulk of the tedious repetitive work vs schematic capture
 - Work at transistor/gate level for large designs: cumbersome
- Portable Design
 - Behavioral or dataflow Verilog can be synthesized to a new process library with little effort (i.e. move from 0.11 μ to 45nm process)

22

Why Use an HDL? (continued)

- Portable Design (continued)
 - Verilog written in ASCII text. The ultimate in portability. Much more portable than the binary files of a GUI schematic capture tool.
- Explore larger solution space
 - Synthesis options can help optimize (power, area, speed)
 - Synthesis options and coding styles can help examine tradeoffs
 - Speed
 - Power
 - area

23

Why Use an HDL? (continued)

- Better Validated Designs
 - Verilog itself is used to create the testbench
 - ✓ Flexible method that allows self checking tests
 - ✓ Unified environment
 - Synthesis tools are very good from the boolean correctness point of view
 - ✓ If you have a logic error in your final design there is a 99.999% chance that error exists in your behavioral code
 - ✓ Errors caused in synthesis fall in the following categories
 - Timing
 - Bad Library definitions
 - Bad coding style...sloppiness

24

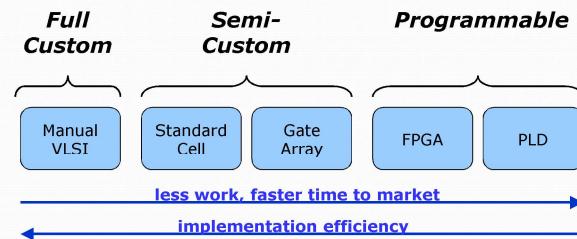
Other Important HDL Features

- Are highly portable (text)
- Are self-documenting (when commented well)
- Describe multiple levels of abstraction
- Represent parallelism
- Provides many descriptive styles
 - Structural
 - Register Transfer Level (RTL)
 - Behavioral
- Serve as input for synthesis tools

25

Hardware Implementations

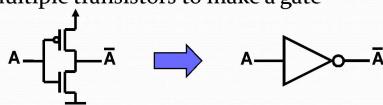
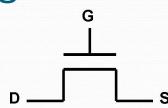
- HDLs can be compiled to semi-custom and programmable hardware implementations



26

Hardware Building Blocks

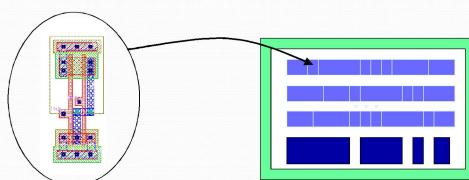
- Transistors are switches
- Use multiple transistors to make a gate
- Use multiple gates to make a circuit



27

Standard Cells

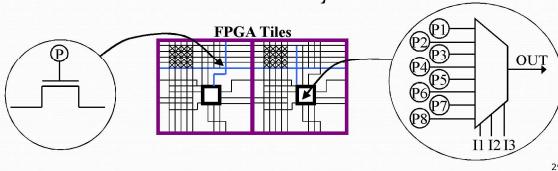
- Library of common gates and structures (cells)
- Decompose hardware in terms of these cells
- Arrange the cells on the chip
- Connect them using metal wiring



28

FPGAs

- “Programmable” hardware
- Use small memories as truth tables of functions
- Decompose circuit into these blocks
- Connect using programmable routing
- SRAM bits control functionality



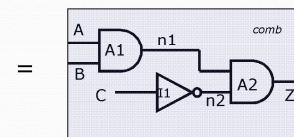
29

What is a Netlist?

- A netlist is an ASCII text representation of the interconnect of a schematic
- Many Standards Exist:
 - Spice Netlist
 - EDIF (Electronic Data Interchange Format)
 - Structural Verilog Netlist (this is what we will use)

```
module comb(Z,A,B,C);
  input A,B,C;
  output Z;
  wire n1, n2;

  and02d1 A1(n1,A,B);
  inv01d1 I1(n2,C);
  and02d1 A2(Z,n1,n2);
endmodule
```



30

Review: Combinational Logic

- Since HDLs try to abstract hardware design, do we even have to consider the hardware?
 - Good hardware design requires ability to analyze a problem to find simplifications
 - Multiple variables: throughput, area, latency, power
 - Finding an optimal hardware implementation is a computationally complex problem. The synthesis tools need guidance on where to start
- Optimization issues
 - if(x != 0) vs. if((x <= -1) || (x >= 1))
 - What hardware might this generate?

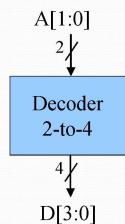
31

Verilog Module

- In Verilog, a circuit is a module.

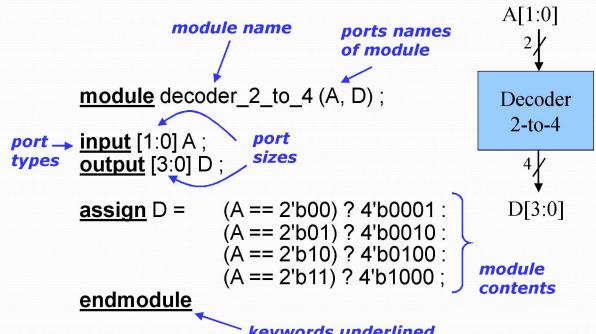
```
module decoder_2_to_4 (A, D);
  input [1:0] A ;
  output [3:0] D ;

  assign D = (A == 2'b00) ? 4'b0001 :
             (A == 2'b01) ? 4'b0010 :
             (A == 2'b10) ? 4'b0100 :
             (A == 2'b11) ? 4'b1000 ;
endmodule
```



32

Verilog Module



33

Declaring A Module

- Can't use keywords as module/port/signal names
 - Choose a *descriptive* module name
- Indicate the ports (connectivity)
- Declare the signals connected to the ports
 - Choose *descriptive* signal names
- Declare any internal signals
- Write the internals of the module (functionality)

34

Good HDL “Self Comments”

```

module xyz123 (A, B);
    input [3:0] A;
    output B;
    //module contents
endmodule

module doomsday_machine (crystals, earthquake);
    input [3:0] crystals;
    output earthquake;
    //module contents
endmodule

```

BAD

GOOD

35 35

Declaring Ports

- A signal is attached to every port
- Declare type of port
 - **input**
 - **output**
 - **inout** (bidirectional)
- Scalar (single bit) - don't specify a size
 - **input** cin;
- Vector (multiple bits) - specify size using range
 - Range is MSB to LSB (left to right)
 - Don't have to include zero if you don't want to... (**D[2:1]**)
 - **output** [7:0] OUT;
 - **input** [0:4] IN;

36

Module Styles

- Modules can be specified different ways
 - Structural – connect primitives and modules
 - Dataflow– use continuous assignments
 - Behavioral – use initial and always blocks
- A single module can use more than one method!
- What are the differences?

37

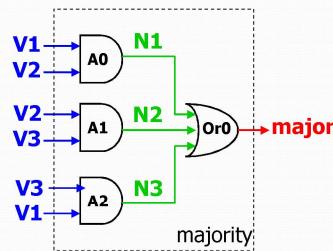
Structural

- A schematic in text form (i.e. A netlist)
- Build up a circuit from gates/flip-flops
 - Gates are primitives (part of the language)
 - Flip-flops themselves described behaviorally
- Structural design
 - Create module interface
 - Instantiate the gates in the circuit
 - Declare the internal wires needed to connect gates
 - Put the names of the wires in the correct port locations of the gates
 - For primitives, outputs always come first

38

Structural Example

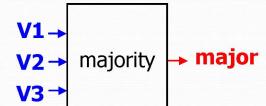
```
module majority (major, V1, V2, V3);  
output major;  
input V1, V2, V3;  
wire N1, N2, N3;  
and A0 (N1, V1, V2),  
       A1 (N2, V2, V3),  
       A2 (N3, V3, V1);  
or Or0 (major, N1, N2, N3);  
endmodule
```



39

RTL Example

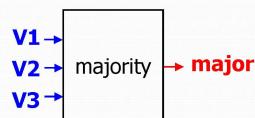
```
module majority (major, V1, V2, V3);  
output major;  
input V1, V2, V3;  
assign major = V1 & V2  
      | V2 & V3  
      | V1 & V3;  
endmodule
```



40

Behavioral Example

```
module majority (major, V1, V2, V3);  
output reg major;  
input V1, V2, V3;  
  
always @ (V1, V2, V3) begin  
    if (V1 && V2 || V2 && V3  
        || V1 && V3) major = 1;  
    else major = 0;  
end  
  
endmodule
```



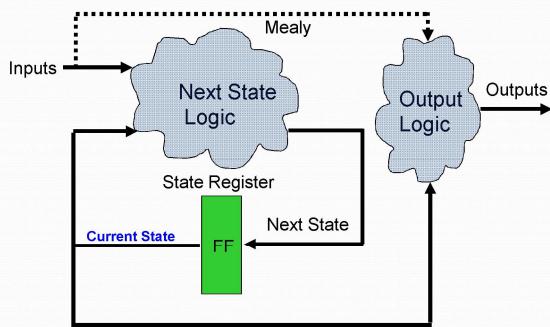
41

FSM Review

- Combinational and sequential logic
- Often used to generate control signals
- Reacts to inputs (including clock signal)
- Can perform multi-cycle operations
- Examples of FSMs
 - Counter
 - Vending machine
 - Traffic light controller
 - Bus Controller
 - Control unit of serial protocol (like RS232, I2C or SPI)

42

Mealy/Moore FSMs



43

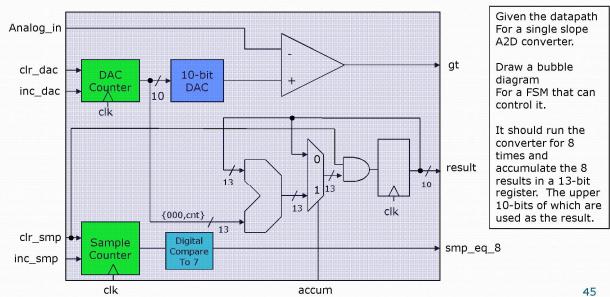
FSMs

- Moore
 - Output depends only on current state
 - Outputs are synchronous (but not necessarily glitch free)
- Mealy
 - Output depends on current state and inputs
 - Outputs can be asynchronous
 - ✓ Change with changes on the inputs
 - Outputs can be synchronous
 - ✓ Register the outputs
 - ✓ Outputs delayed by one cycle

44

Remember Bubble Diagrams?

- They can be useful. I sometimes will draw a bubble diagram first for a complex FSM. Then code it.



45

Things to do

- Familiarize self with the Moodle webpage
- Look at the IEEE Verilog standard on webpage

46

Review Questions

- What are some advantages of using HDLs, instead of schematic capture?
- What advantages and disadvantages do standard cell designs have compared to full-custom designs?
- What are some ways in which HDLs differ from conventional programming languages? How are they similar?
- What are the different styles of Verilog coding?

47