

ET5080E

Digital Design Using Verilog HDL

Fall '21

Lecture 1

- Instructor Introduction
- Course Introduction
- Administrative Stuff
- Introduction to Verilog

Instructors

Vu-Duc Ngo, Graduated in HUST,1997

- Ph.D in KAIST
- duc.ngovu@hust.edu.vn
- Research interest: Digital IC design for Wireless Comms, Multimedia Codec, NoC, Embedded Systems.

Course Goals

- Provide knowledge and experience in:
 - Digital circuit design using a HDL (Verilog)
 - HDL simulation
 - How to build self checking test benches
 - Good practices in digital design verification
 - Synthesis of dataflow and behavioral designs
 - Basic static timing analysis concepts
 - Optimizing hardware designs (timing, area, power)
 - Design tools commonly used in industry

- Teach you to be able to “think hardware”

What You Should Already Know

- Principles of basic digital logic design
- Number representations (unsigned, signed, Hex & Binary)
 - Boolean algebra
 - Gate-level design
 - K-Map minimization
 - Finite State Machines
 - Basic datapath structures (adders, shifters, SRAM)

Course Materials

- Lectures
- Textbook
 - Samir Palnitkar, Verilog HDL, Prentice Hall, 2003.
- 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
- Xilinx or Synopsys on-line documentation
- Other useful readings

Evaluation and Grading

- Approximately:
 - 30% Midterm as Term project
 - 70% Final

- Homework due at beginning of class
 - **15% penalty** for each late period of 24 hours
 - Not accepted >72 hours after deadline
 - *Your* responsibility to get it to me
 - ✓ Can leave in my mailbox with a timestamp of when it was turned in

Class Project

- Work in groups of 2 or 3 students
- Design, model, simulate, synthesize, and test a complex digital system
- Several milestones
 - Forming teams
 - Project status report
 - Progress demonstrations (maybe)
 - Final demonstration & report
- More details coming later in the course

Course Tools

- Industry-standard design tools:
 - Modelsim HDL Simulation Tools (Mentor)
 - Design Vision Synthesis Tools (Xilinx)

- Tutorials will be available for both tools
 - Modelsim tutorial next week
 - Design Vision tutorial a few weeks later
 - Tool knowledge will be required to complete homeworks

ModelSim Tutorial

- ModelSim is our Verilog simulator.
 - Check out the tutorial on Youtube

Readings for Week 1

- Read Chapter 1 (short)
 - Overview of Digital Design with Verilog HDL
- Read Chapter 2 (short)
 - Hierarchical Modeling Concepts
- Read Chapter 3
 - Basic Verilog Syntax

What is an HDL?

- HDL = **H**ardware **D**escription **L**anguage
 - 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
 - ✓ Once you have the concept of an HDL down (can think and code hardware), the language makes little difference.

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
```

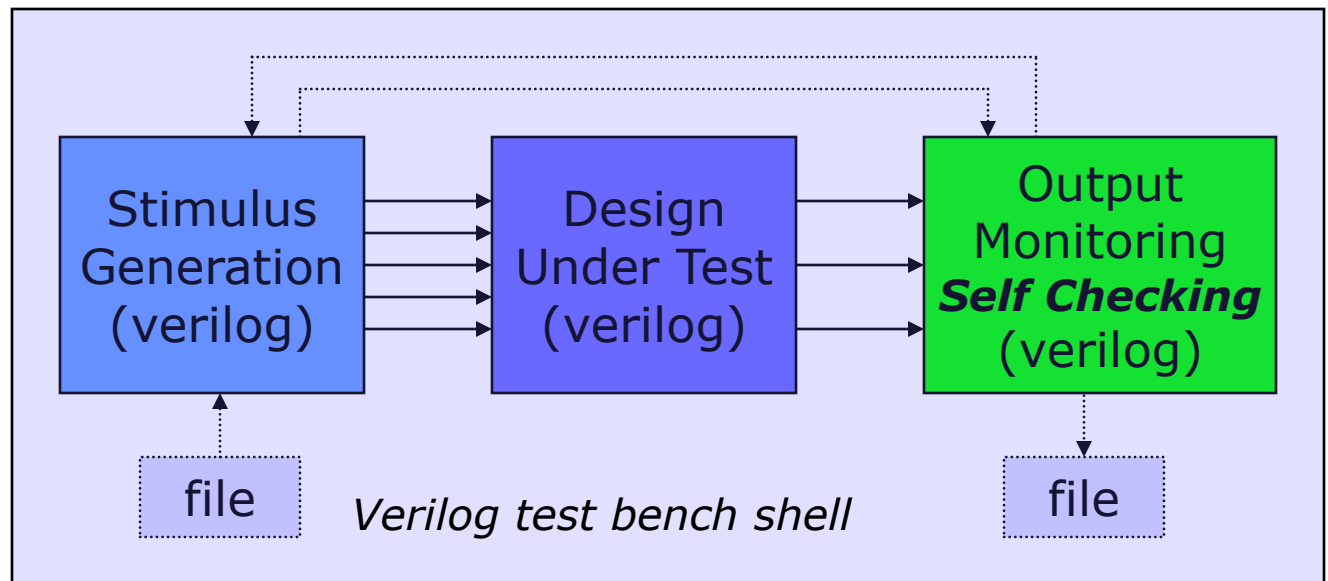
- It looks like a programming language
- It is **NOT** a programming language
 - ✓ It is always critical to recall you are describing hardware
 - ✓ This codes primary purpose is to generate hardware
 - ✓ The hardware this code describes (a counter) can be simulated on a computer. In this secondary use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

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.

Testbenches are written in verilog as well.

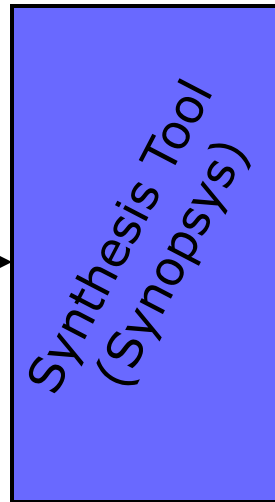
Testbench verilog is not describing hardware and can be thought of as more of a program.



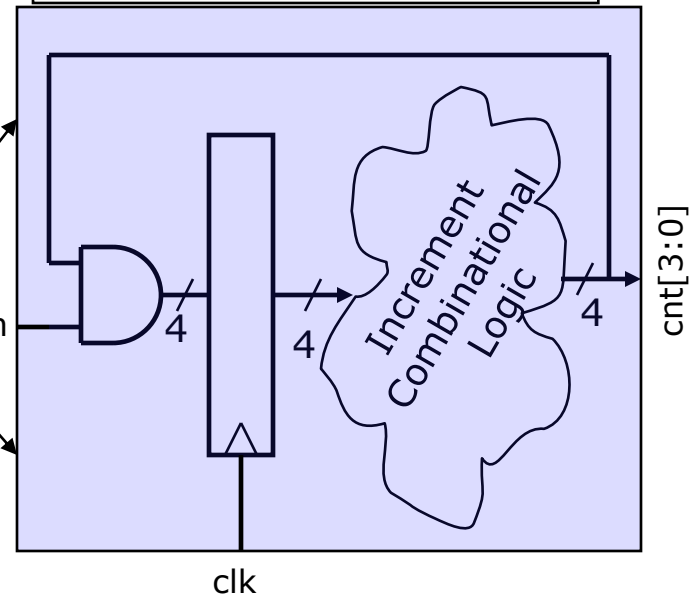
What is Synthesis?

- 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
```



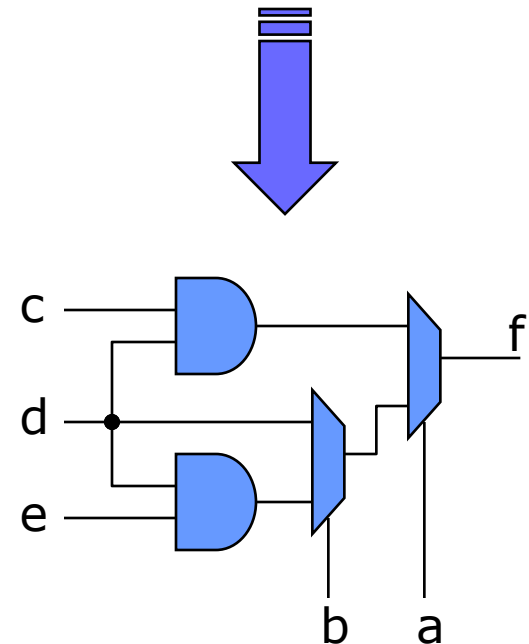
Output is actually a text netlist, not a GUI schematic form.



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;
```



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)

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

Why Use an HDL? (continued)

■ Better Validated Designs

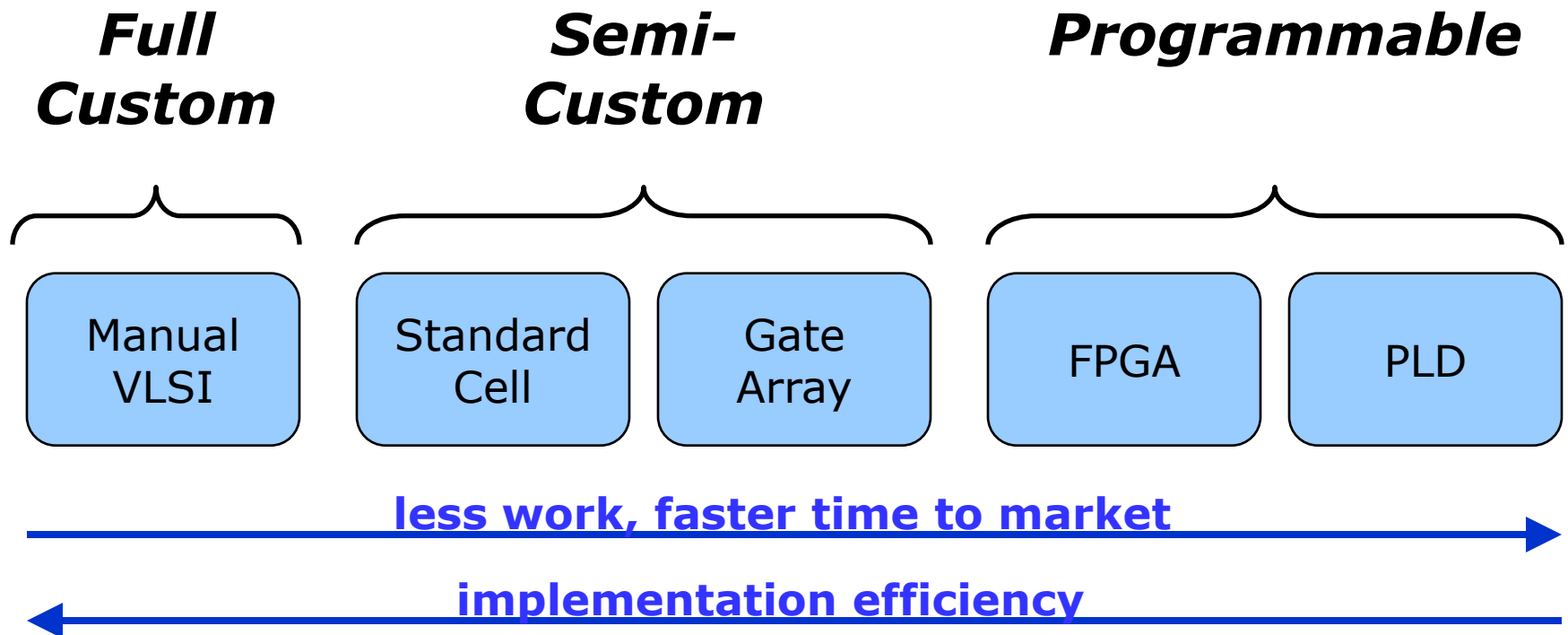
- Verilog itself is used to create the testbench
 - ✓ Flexible method that allows self checking tests
 - ✓ Unified environment
- Synthesis tool are very good from the boolean correctness point of view
 - ✓ If you have a logic error in your final design there is a 99.999% chance that error exists in your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

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

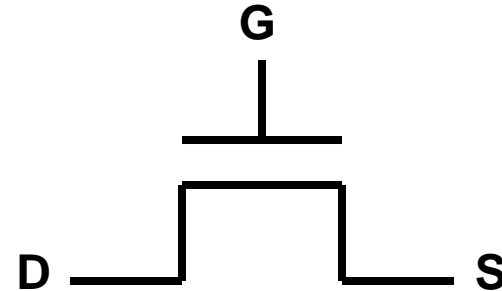
Hardware Implementations

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

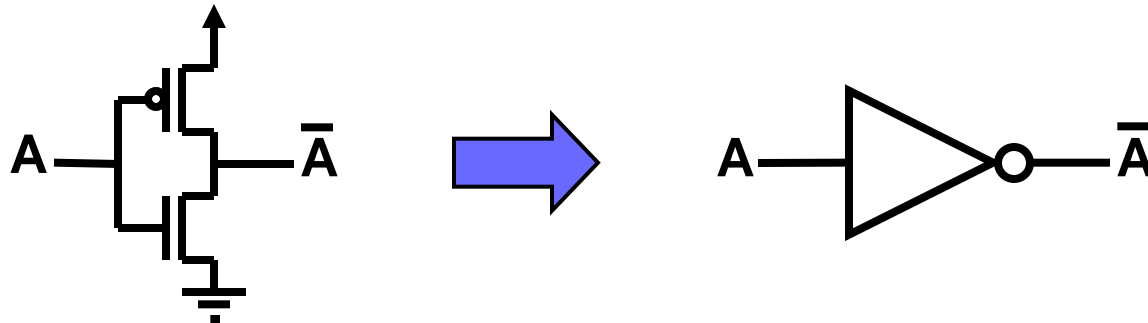


Hardware Building Blocks

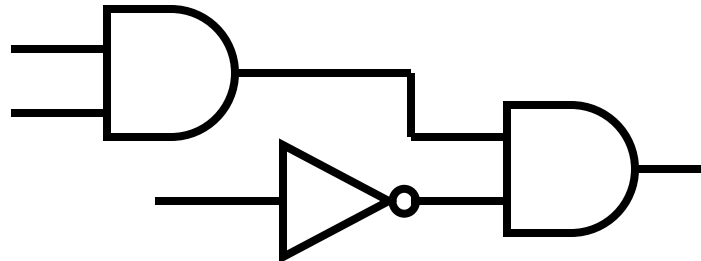
- Transistors are switches



- Use multiple transistors to make a gate

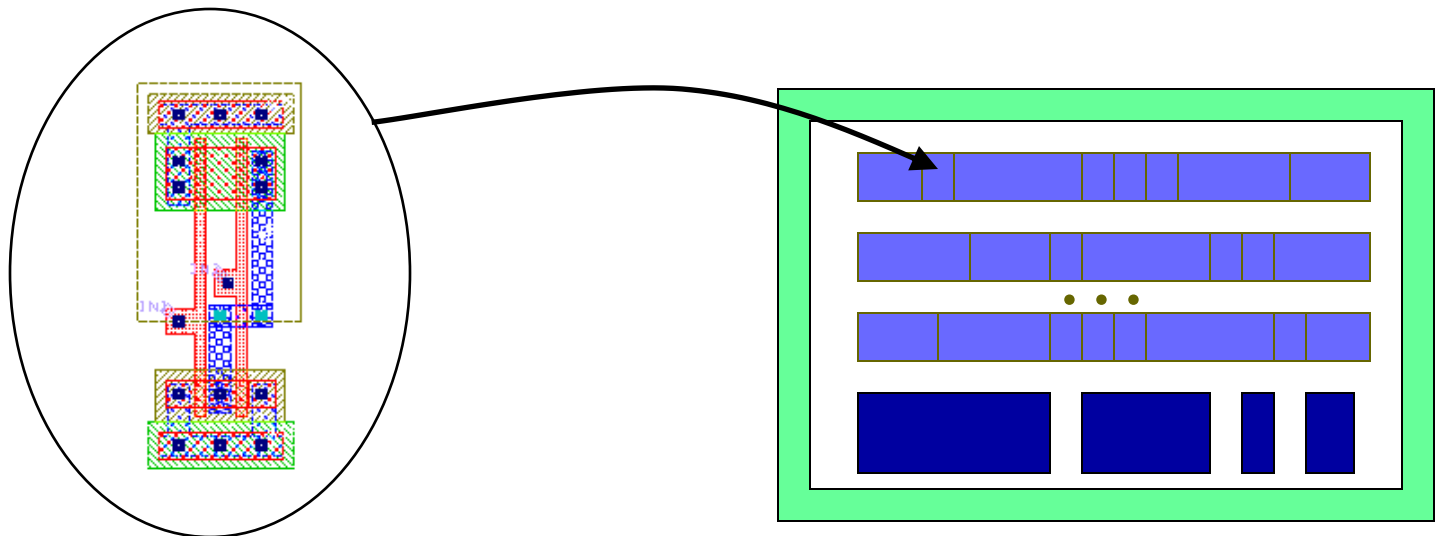


- Use multiple gates to make a circuit



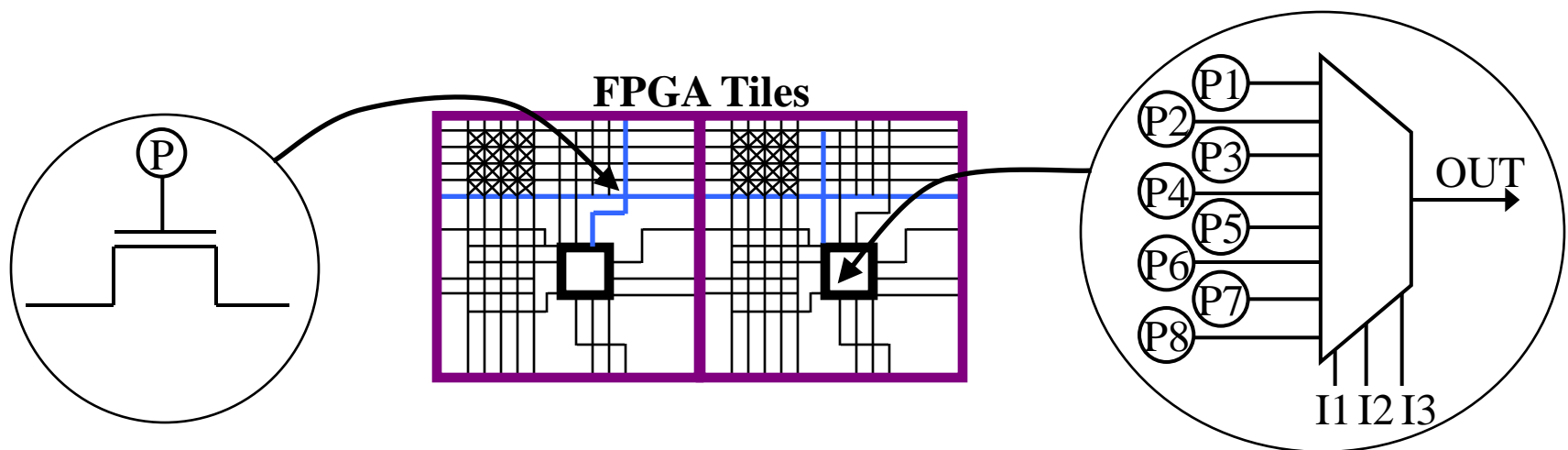
Standard Cells

- 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



FPGAs

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

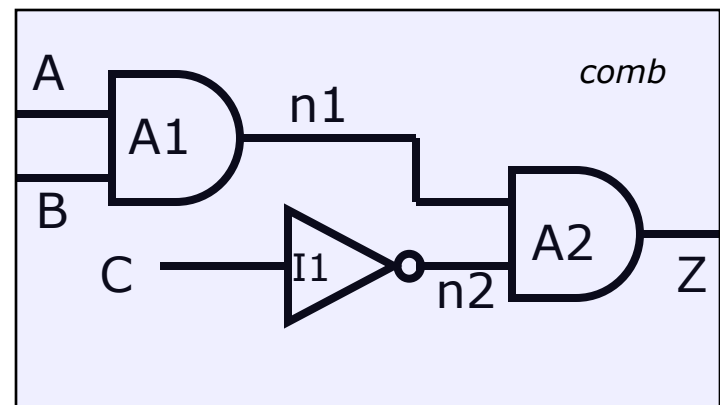


What is a Netlist?

- A netlist is a 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
```

=

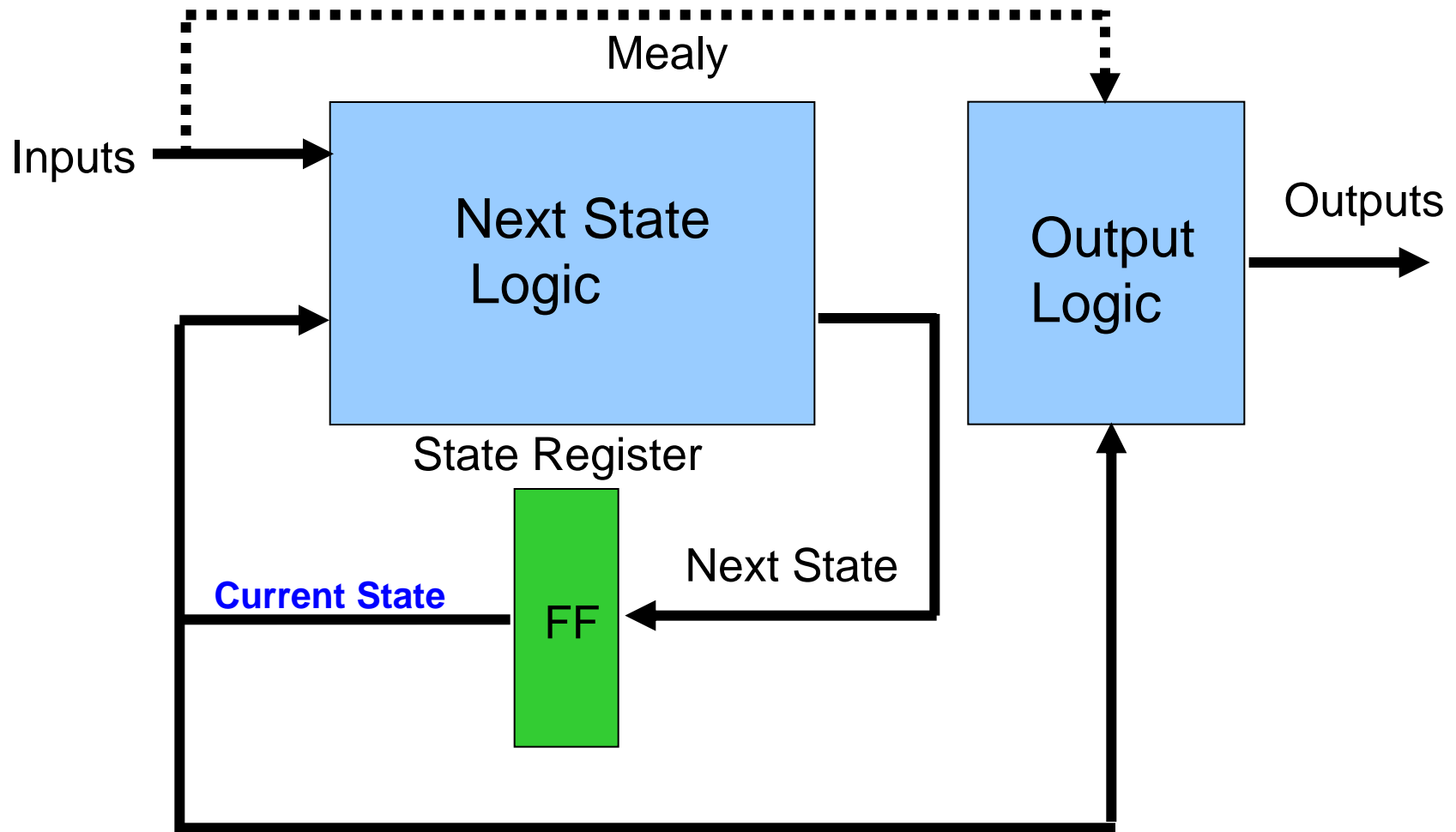


FSM Review

- 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)

Mealy/Moore FSMs



FSMs

■ Moore

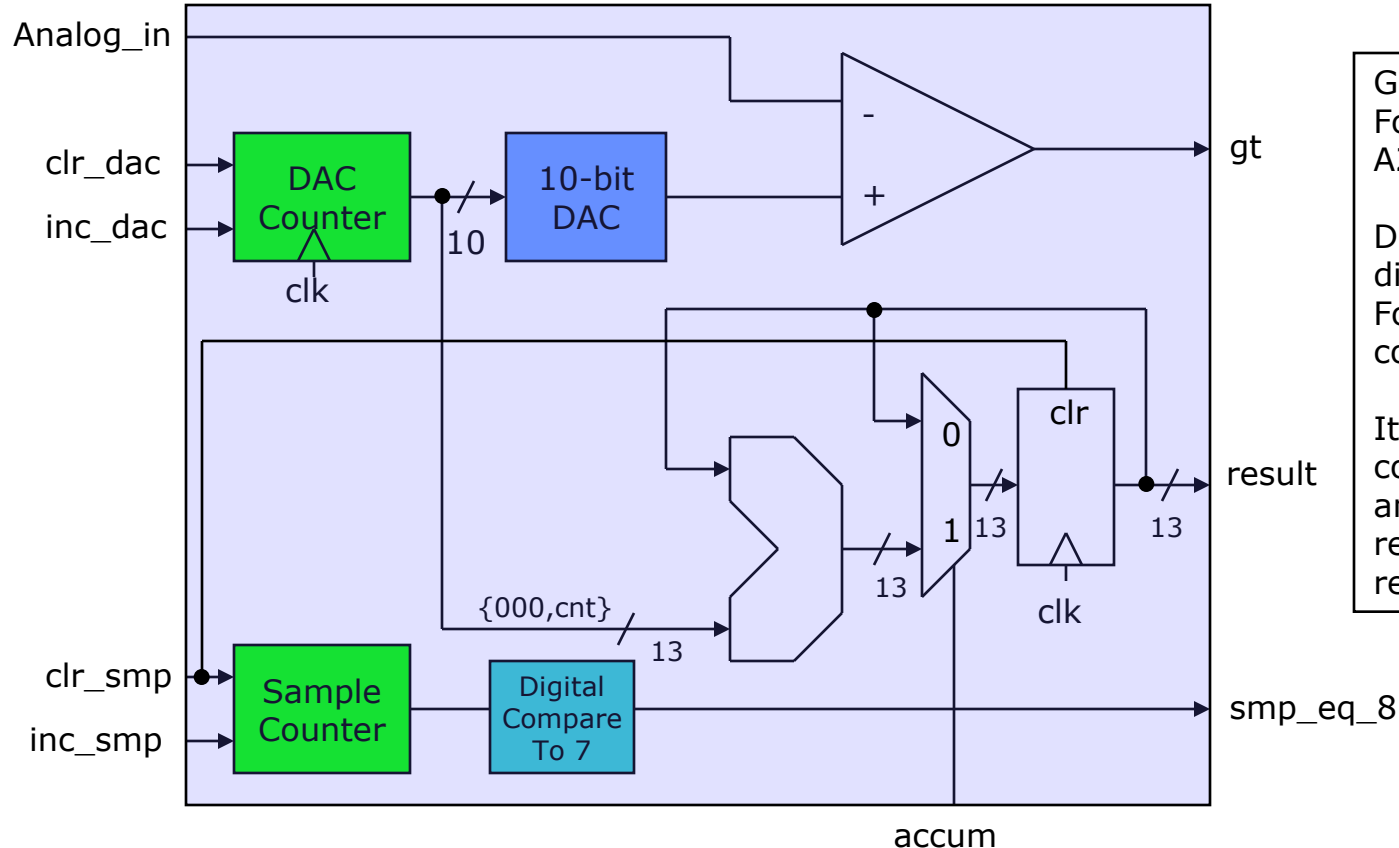
- Output depends only on current state
- Outputs are synchronous (but not necessarily glitch free)

■ 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

Remember Bubble Diagrams?

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



Given the datapath
For a single slope
A2D converter.

Draw a bubble
diagram
For a FSM that can
control it.

It should run the
converter for 8 times
and accumulate the 8
results in a 13-bit
register.

Verilog Module

- In Verilog, a circuit is a module.

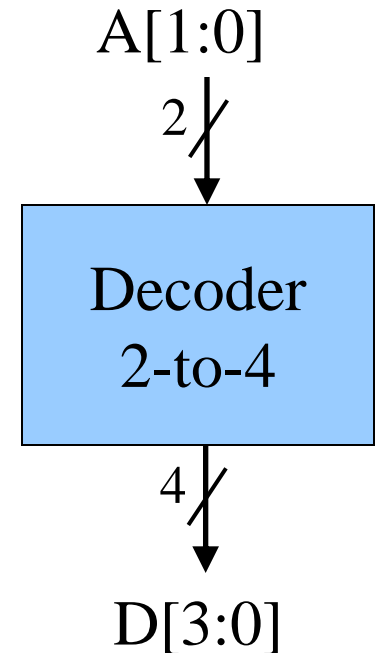
```
module decoder_2_to_4 (A, D) ;
```

```
input [1:0] A ;
```

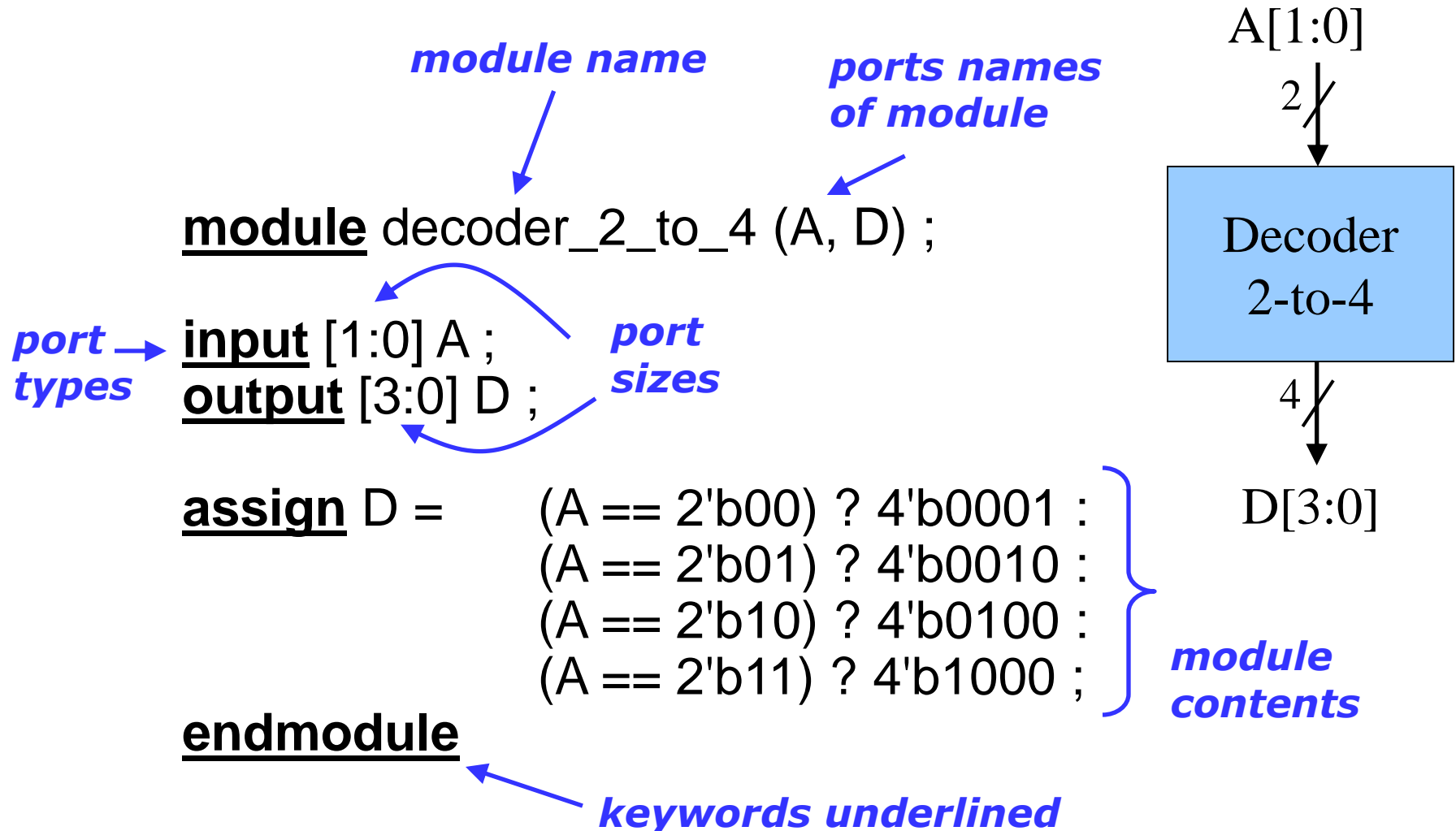
```
output [3:0] D ;
```

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

```
endmodule
```



Verilog Module



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)

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;

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?

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

Structural Example

```
module majority (major, V1, V2, V3) ;
```

```
output major ;
```

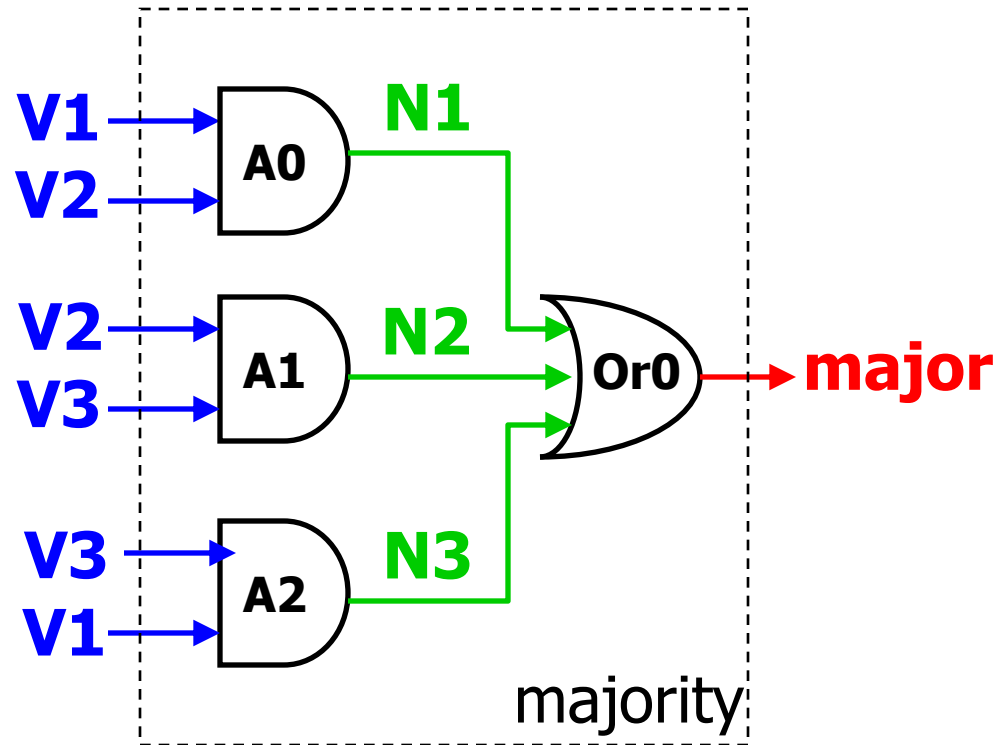
```
input V1, V2, V3 ;
```

```
wire N1, N2, N3;
```

```
and A0 (N1, V1, V2),  
    A1 (N2, V2, V3),  
    A2 (N3, V3, V1);
```

```
or Or0 (major, N1, N2, N3);
```

```
endmodule
```



RTL Example

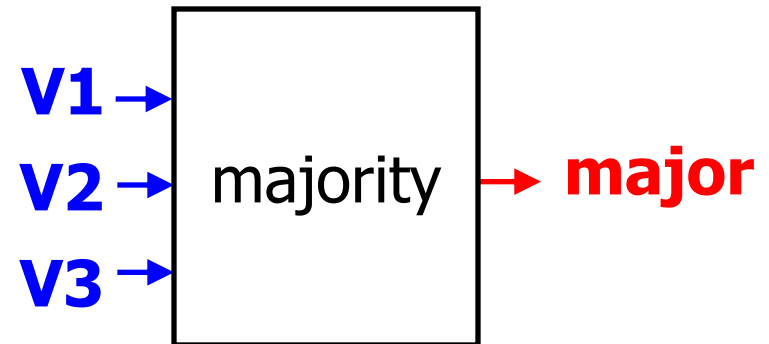
```
module majority (major, V1, V2, V3) ;
```

```
output major ;
```

```
input V1, V2, V3 ;
```

```
assign major = V1 & V2  
          | V2 & V3  
          | V1 & V3;
```

```
endmodule
```



Behavioral Example

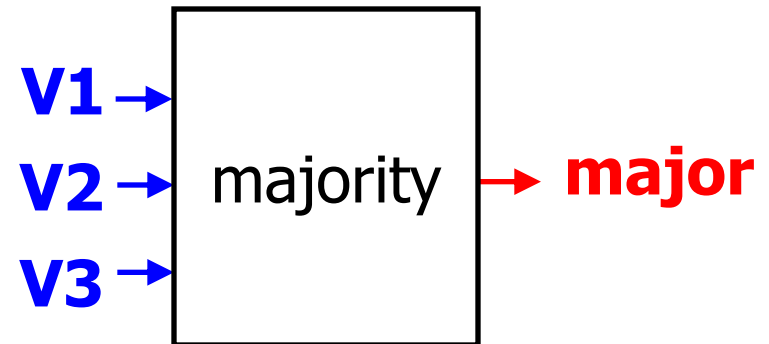
```
module majority (major, V1, V2, V3) ;
```

```
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
```



Things to do

- Read Chapter 1 (short)
 - Overview of Digital Design with Verilog HDL
- Read Chapter 2 (short)
 - Hierarchical Modeling Concepts
- Read Chapter 3
 - Basic Verilog Syntax (a little dry/boring but necessary)
- Familiarize self with eCOW webpage

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?