Digital Design Process

Specification	Planning	Design Entry	Functional Test

Synthesis	Post-Synthesis Test	APR, Parasitic Extraction & Timing Checks	Manufacture & Hardware Validation

2

)_ HOL

Description Phase

Specification	Planning	Design Entry	Functional Test

Implementation Phase

Synthesis	Post-Synthesis Test	APR, Parasitic Extraction & Timing Checks	Manufacture & Hardware Validation

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
    - ✓ System Verilog (Sverilog) is the next generation of verilog, and we will be learning portions of system verilog as well.

What is an HDL? (continued)

```verilog
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 code's 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.


Verilog test bench shell
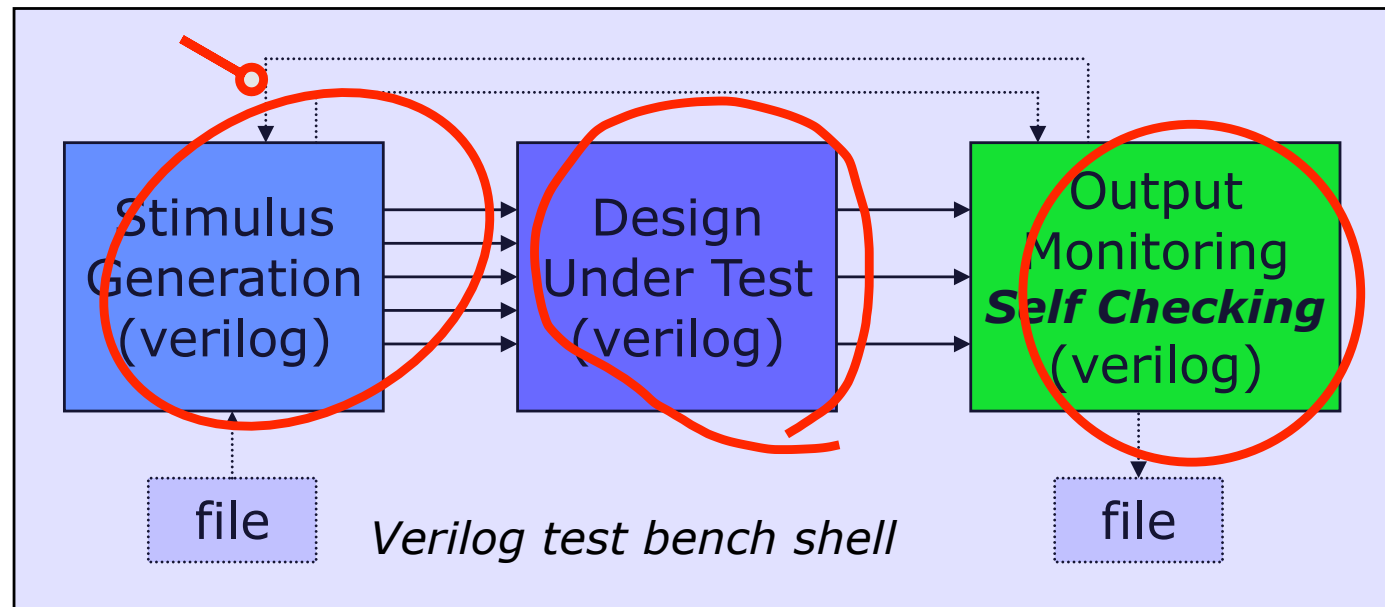
What is Synthesis?

- 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
```
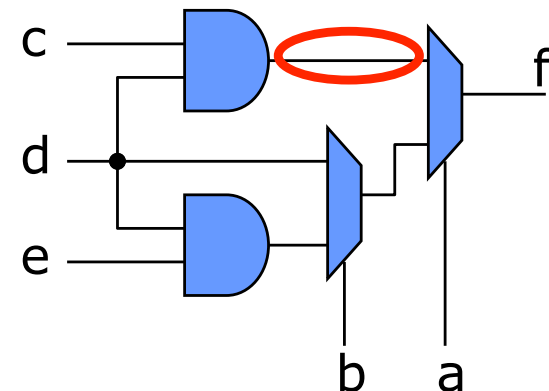
Synthesis Tool
(Synopsys)

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

rst_n

4

4

Increment Combinational Logic

4

cnt[3:0]

clk
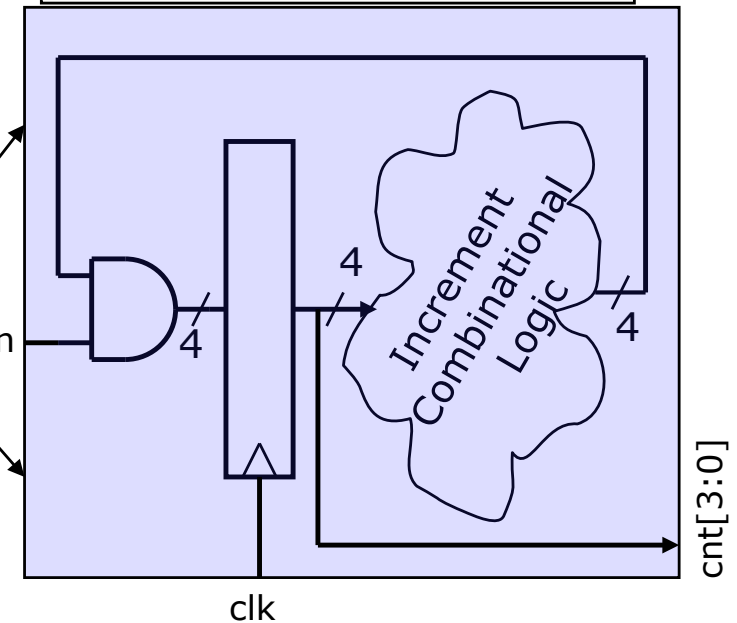
Synthesizing the Hardware Described

- All hardware created during synthesis
  - Even if `a` is true, still computing `d&e`

- Learn to understand how descriptions are translated to hardware

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

Why Use an HDL?

- Enables Larger Designs via rich syntax, modularity
  - More abstract 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 0.11µ 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

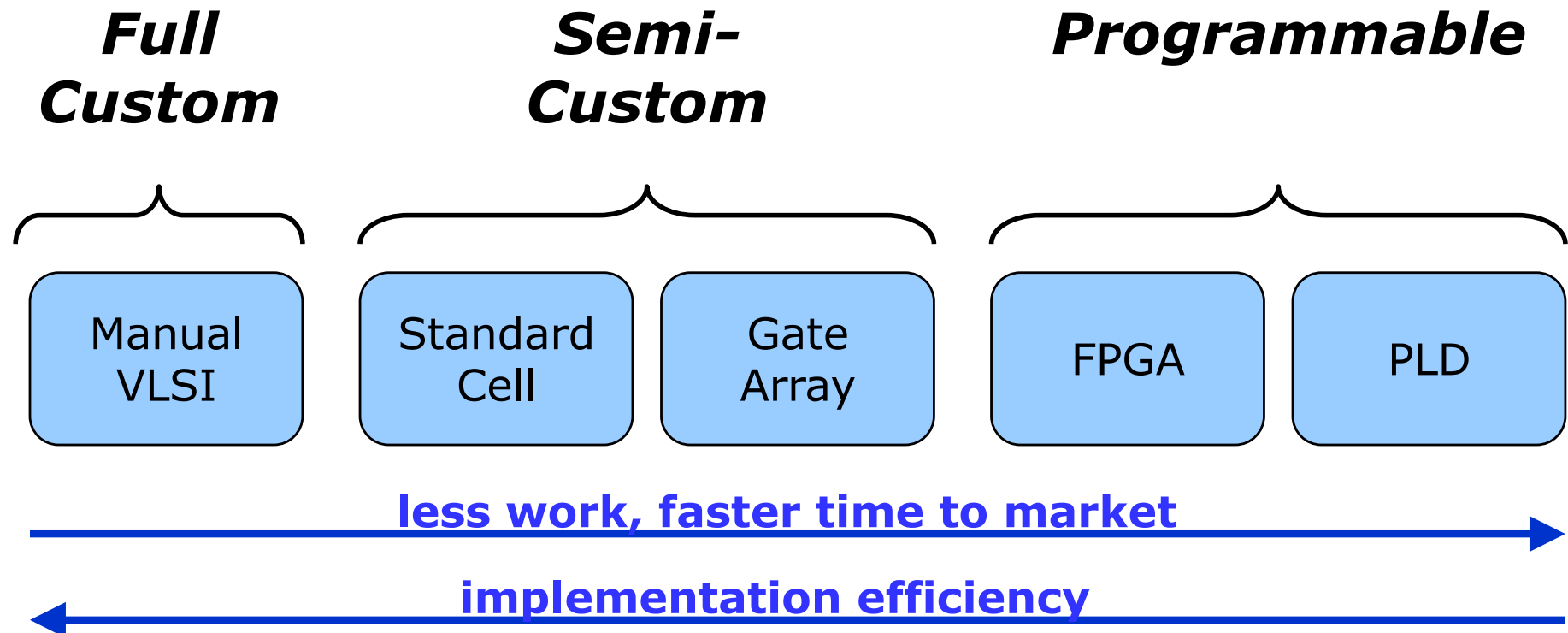# Why Use an HDL? (continued)

- Better Validated Designs
  - Verilog itself is used to create the testbench
    - ✓ Flexible method that allows self checking tests
    - ✓ Unified environment
  - Synthesis 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…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)
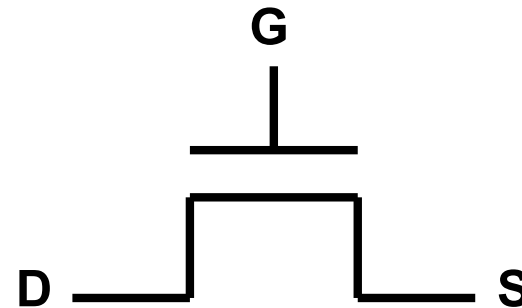 - Behavioral
- Serve as input for synthesis tools

Hardware Implementations

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

Full Custom   **Semi-Custom**   **Programmable**

| Manual VLSI | Standard Cell | Gate Array | FPGA | PLD |

less work, faster time to market →
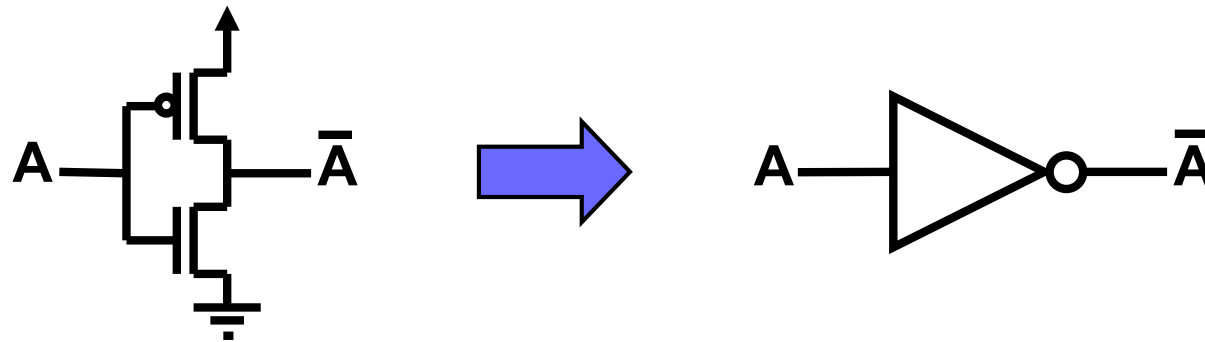
← **implementation efficiency**
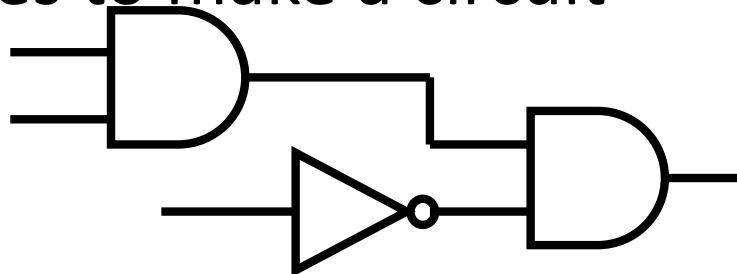
13

Hardware Building Blocks

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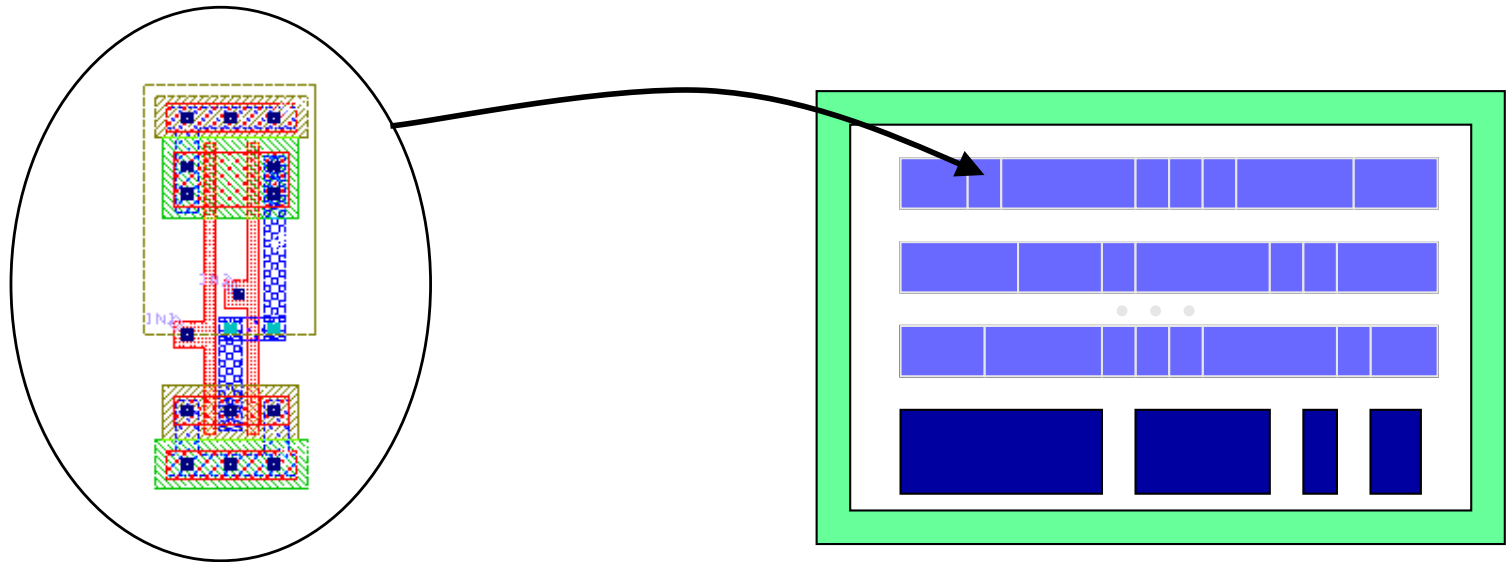
- Use multiple transistors to make a gate

- Use multiple gates to make a circuit

14

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

FPGA Tiles
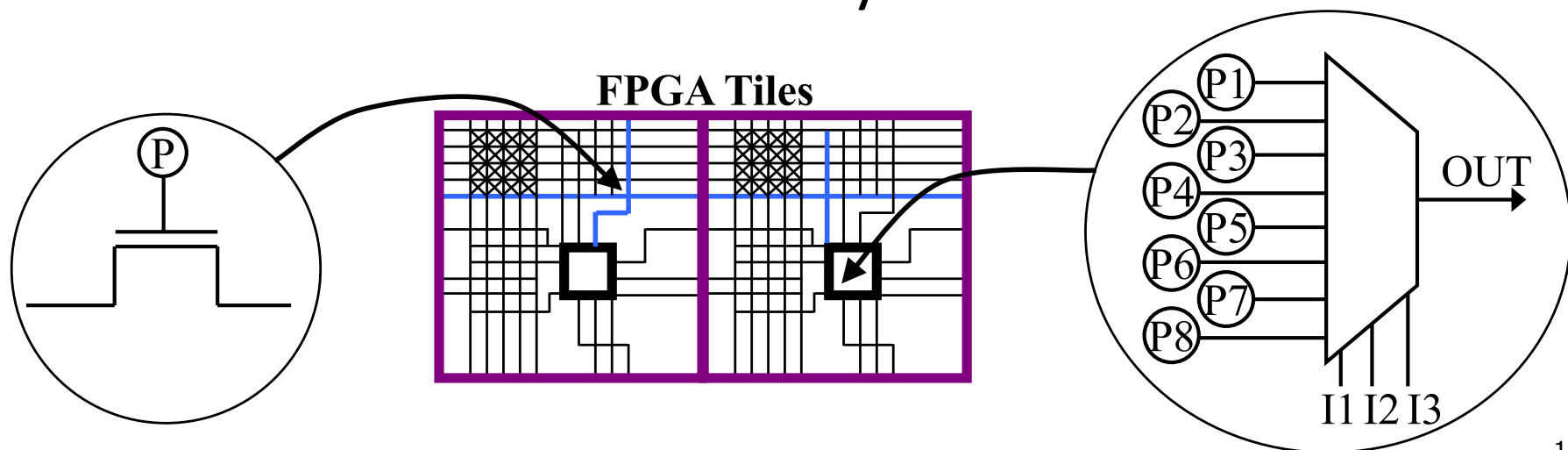
What is a Netlist?

- 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
```

=
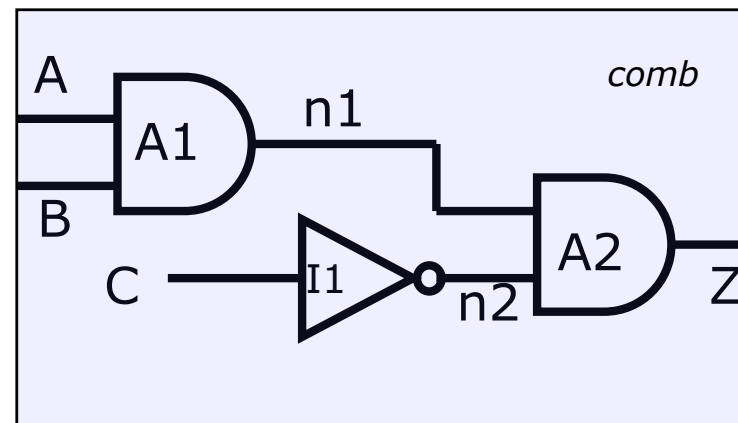
Review: Combinational Logic

- 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?

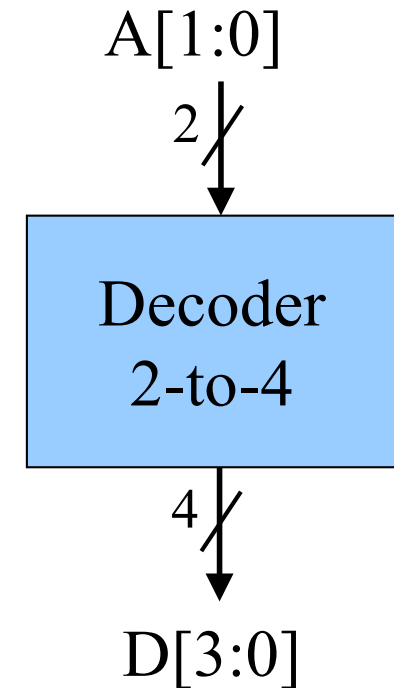# Verilog Module

- In Verilog, a circuit is a <u>module</u>.

  module decoder_2_to_4 (A, D) ;

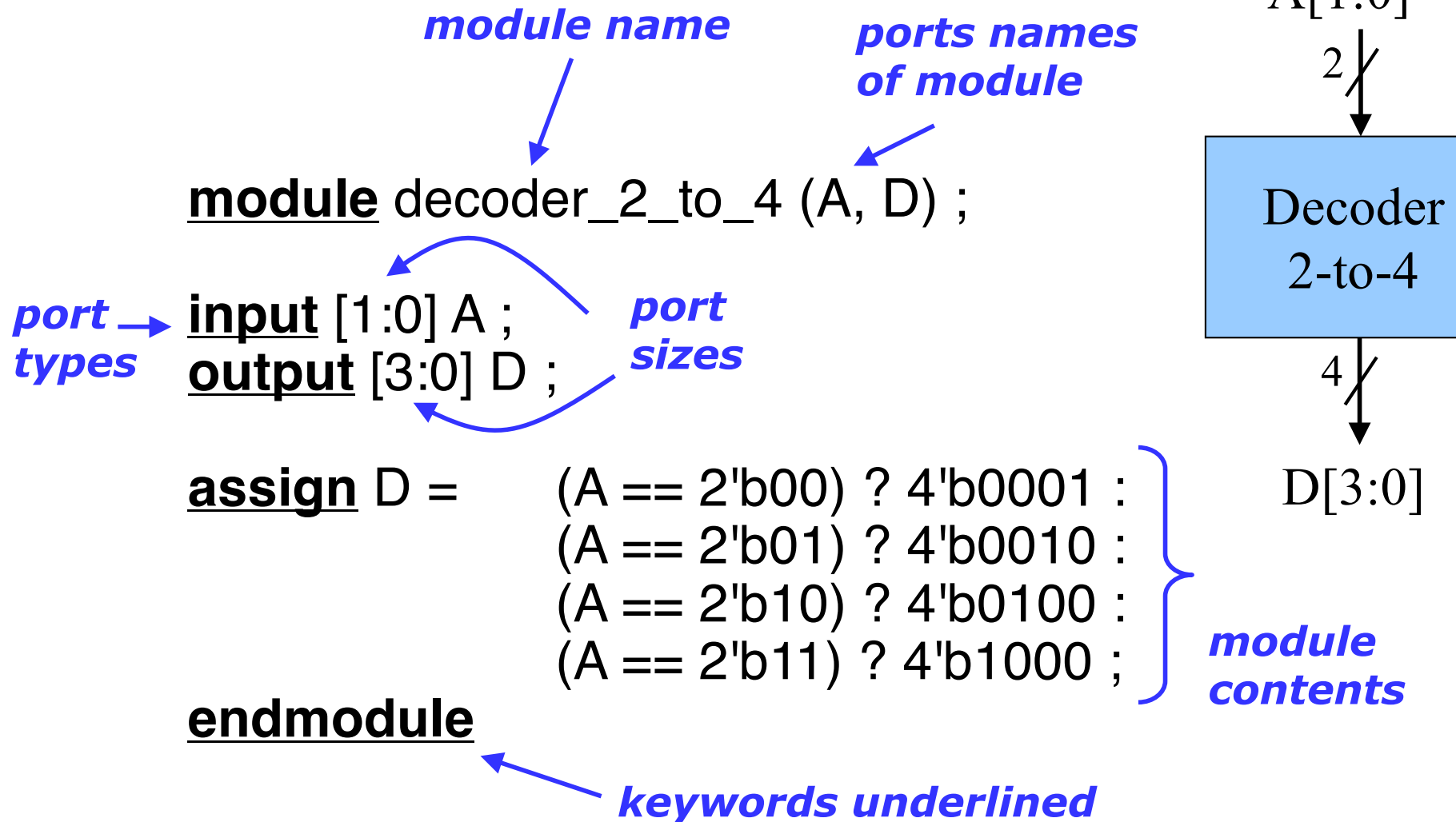 **input** [1:0] A ;
  output [3:0] D ;

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

$A[1:0]$

2

Decoder
2-to-4

4

$D[3:0]$

19

Verilog Module

module name

ports names
of module

A[1:0]

2

Decoder
2-to-4

module decoder_2_to_4 (A, D) ;

port
types

input [1:0] A ;
output [3:0] D ;

port
sizes

4

D[3:0]

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

module
contents

endmodule

keywords underlined

20

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)

Good HDL "Self Comments"

```verilog
module xyz123 (A, B) ;
        input [3:0] A ;
        output B ;

        //module contents
endmodule
```

BAD

```verilog
module doomsday_machine (crystals, earthquake) ;
        input [3:0] crystals ;
        output earthquake ;

        //module contents
endmodule
```

GOOD

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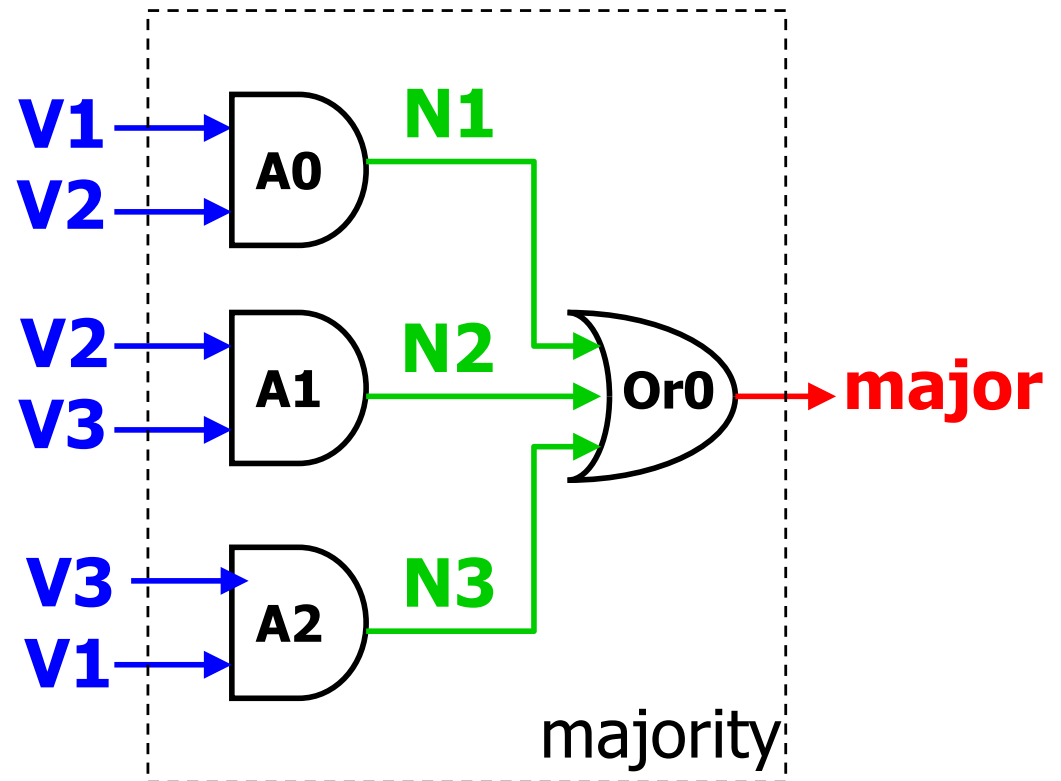
module majority (major, V1, V2, V3) ;

output major ;
input V1, V2, V3 ;

wire N1, N2, N3;

and A0 (N1, V1, V2),
    A1 (N2, V2, V3),
    A2 (N3, V3, V1);
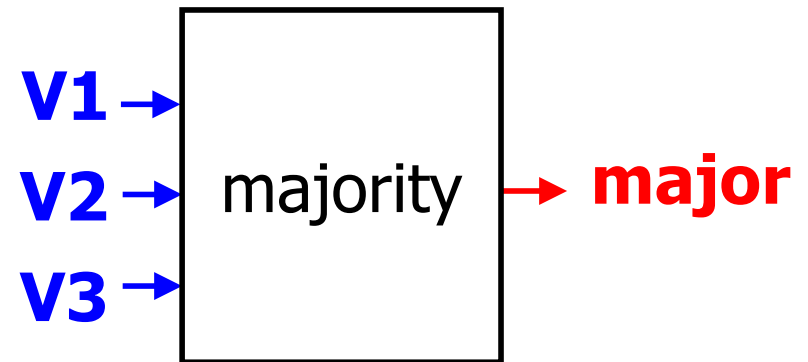
or Or0 (major, N1, N2, N3);

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;
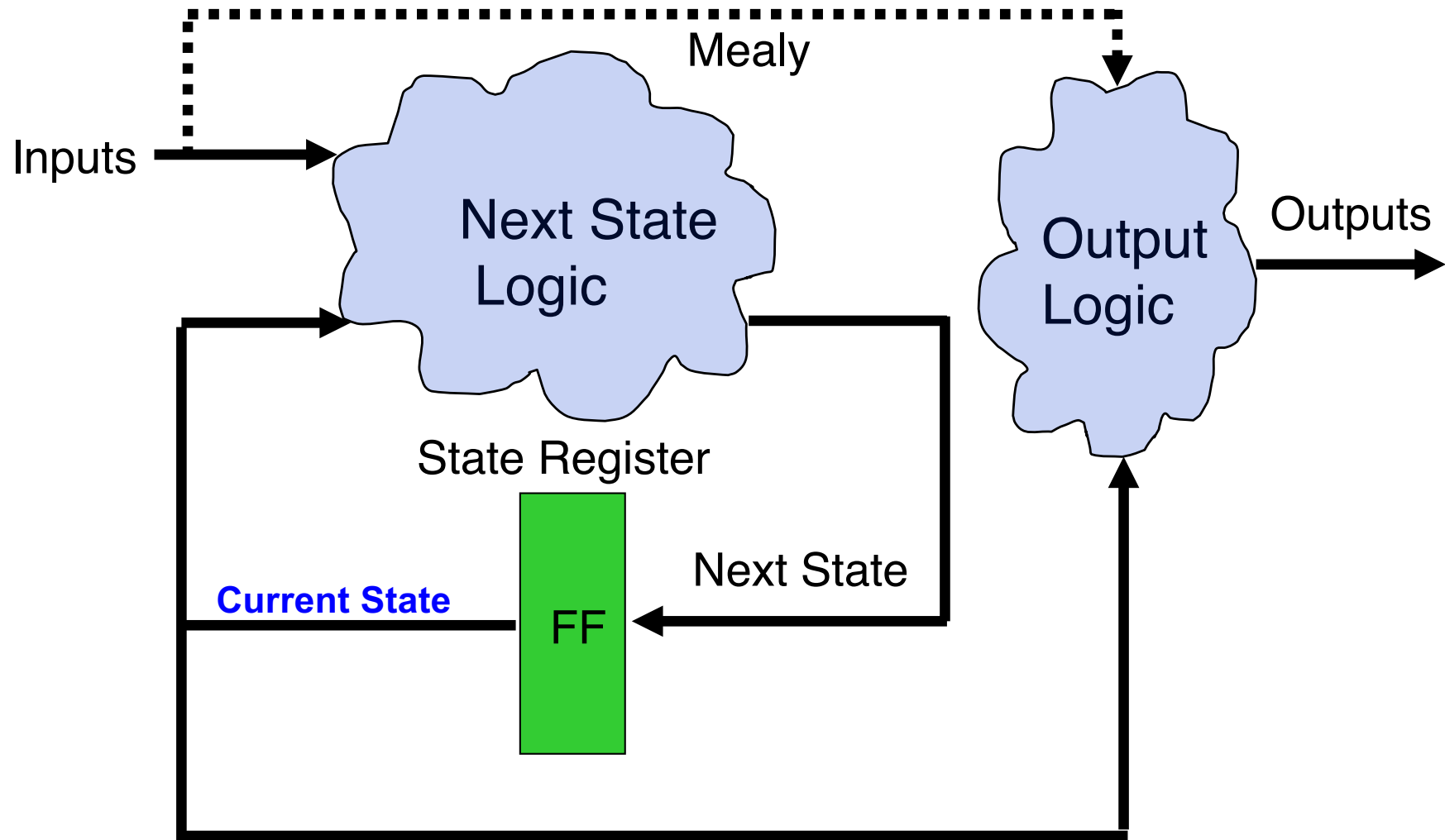   **else** major = 0;
end

endmodule
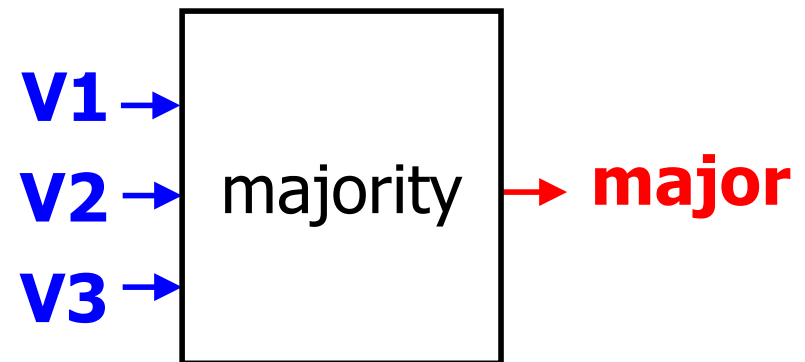
FSM Review

- 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

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?