CSE 260M / ESE 260
Intro. To Digital Logic & Computer Design

Bill Siever
&
Jim Feher

This week

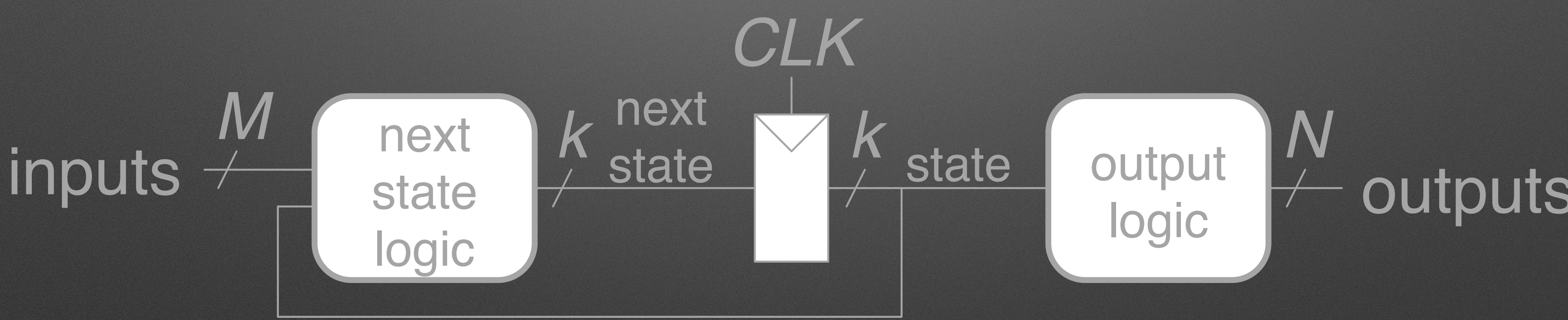
- Thursday:
  Studio — Watch Piazza/Email for location change.
  Expected to be 3rd floor Seigle Hall

- Expecting to return Hw 2 and 3 by next week

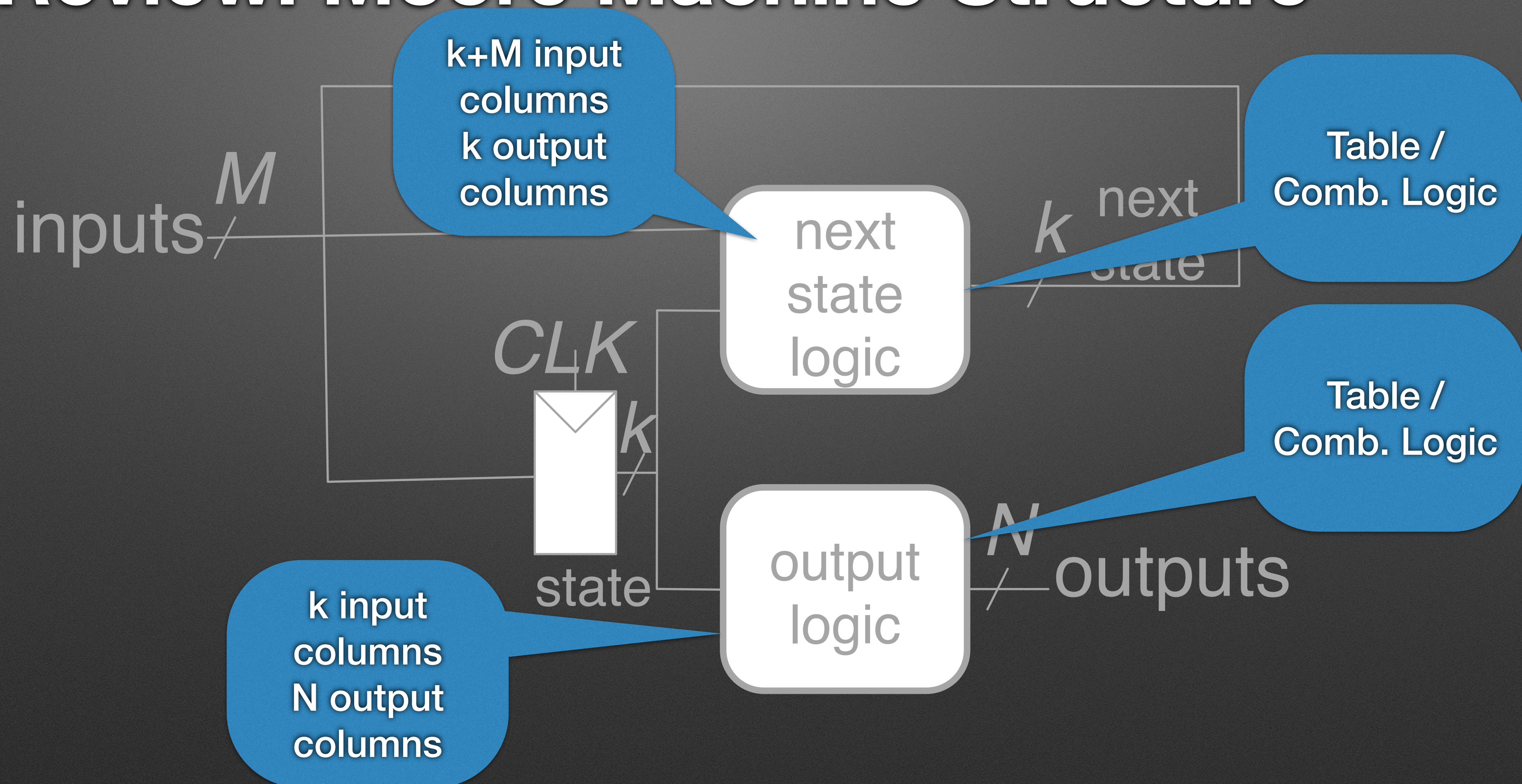
Next week

- Mon/Tues: Fall break
- Hw #4 Due Wednesday - Don’t wait until the last minute!!!
  - Hopefully < 3 hours work; All review and good test prep!
  - Autograde feedback in Gradescope to check your work / resubmit
- Exam 1: Thursday during class
  - Exam 1 page updated with some details

Review: Moore Machine Structure



Review: Moore Machine Structure



Studio Review

- SR-Latch: Hazardous conditions ahead!

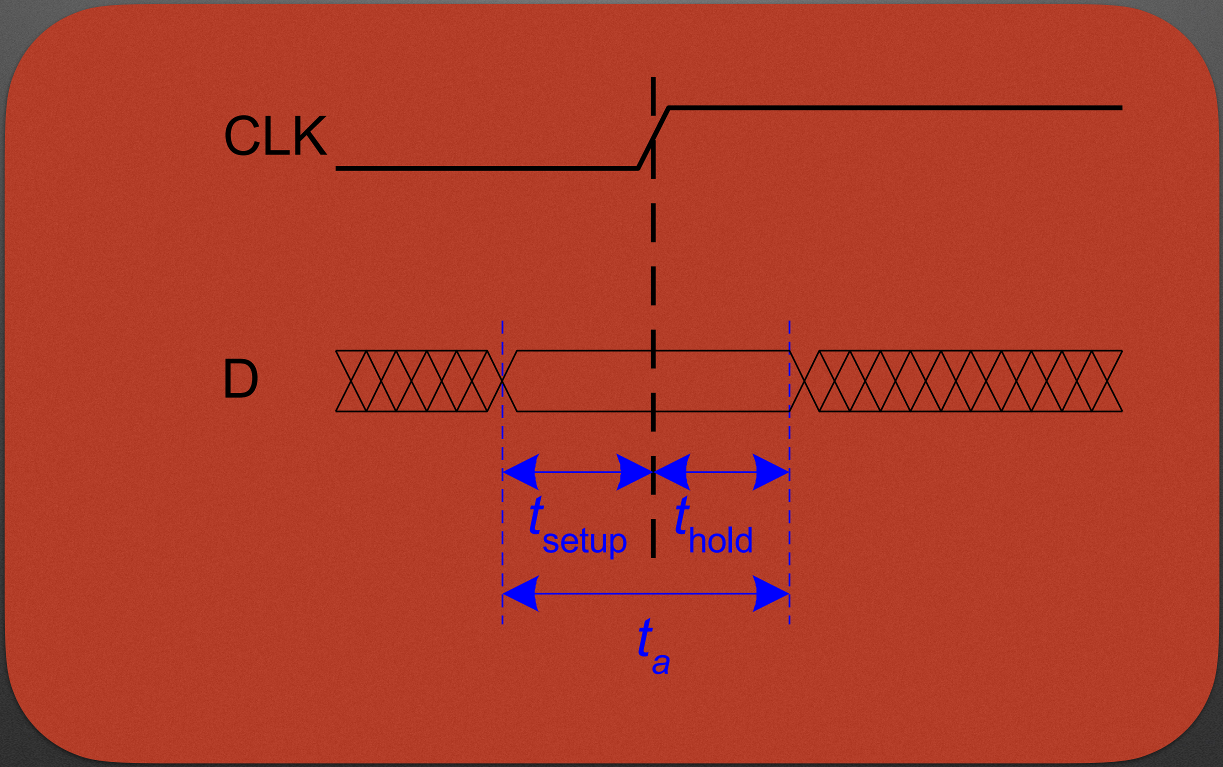
Studio Review

- D-Flip Flop: First Gate Behavior
  - Provoking the SR Latch to fail by moving D around
  - We’ll “watch” things of interest
    - Clock, D, nClock, nD, q1
    - And add “probes” to reset/set lines

D Flip Flop Time Parameters

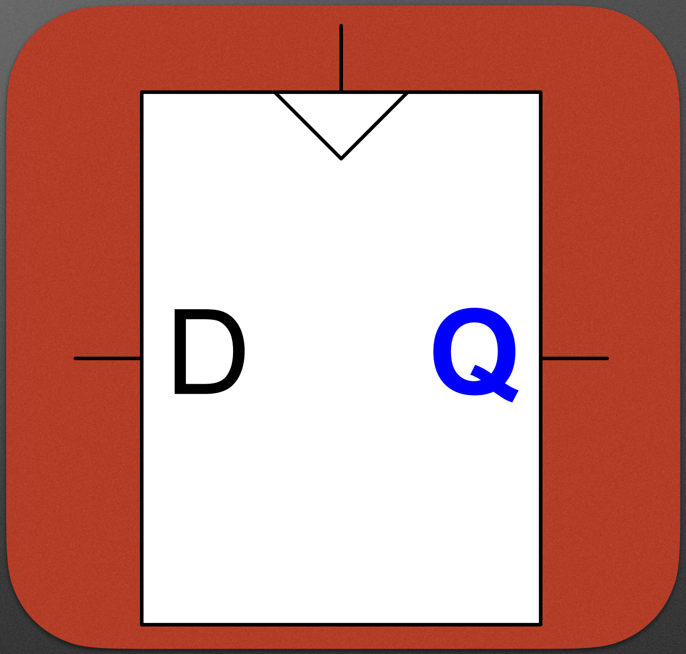
- t_{setup}: Time D must be stable before clock
- t_{hold}: Time D must be stable after clock
- t_a: Aperture time ($t_{setup} + t_{hold}$) –
  total window of time D needs to be stable around clock

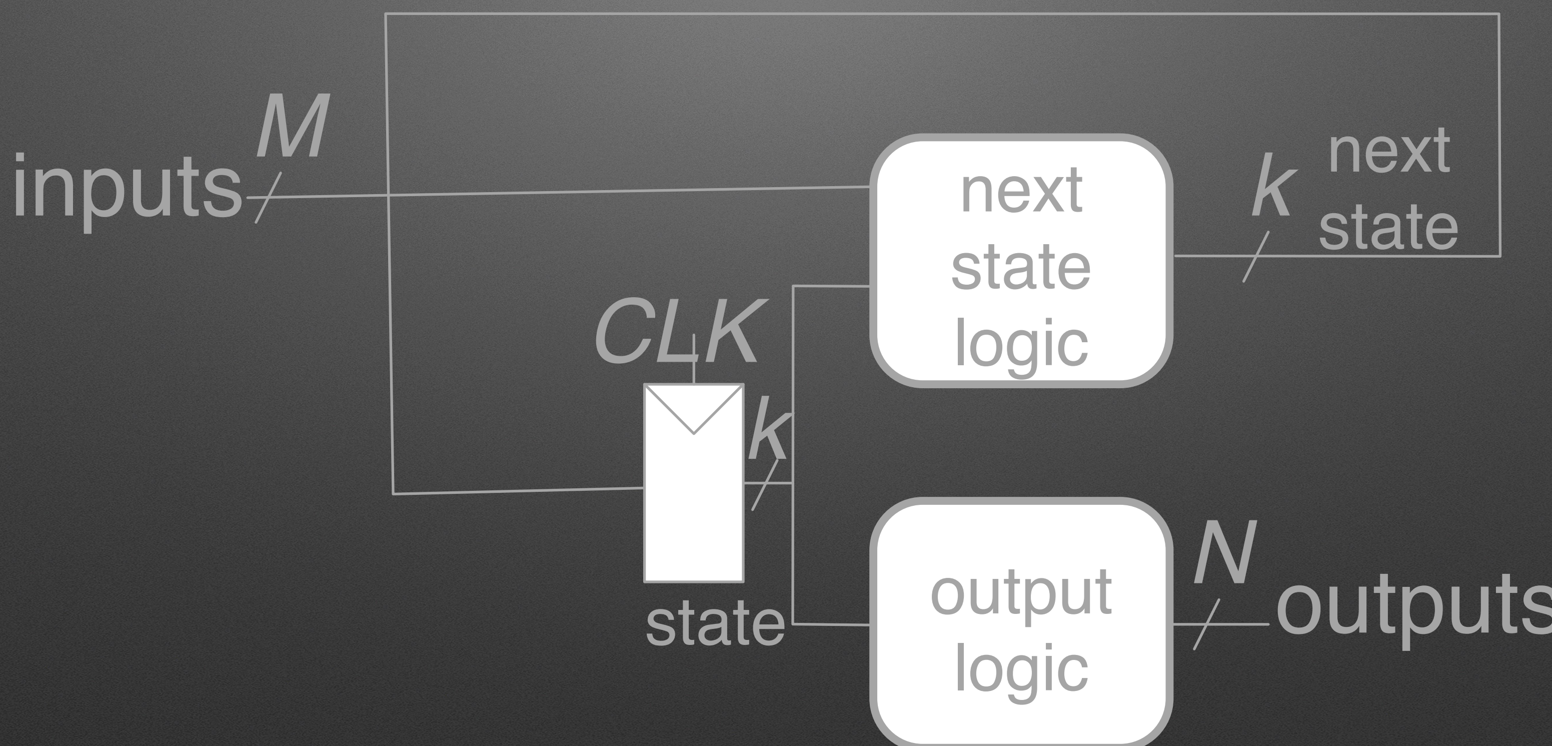
Dff: Setup & Hold Time



Dff Time Parameters

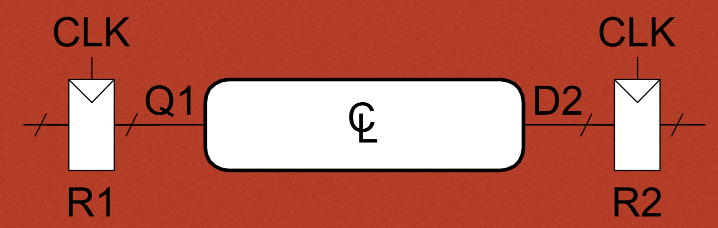
- t_{pcq}: Propagation delay from Clock to Q (pcq)
- t_{ccq}: Contamination delay from C to Q (ccq)




inputs
M
next state logic
k
next state
CLK
k
state
output logic
N
outputs

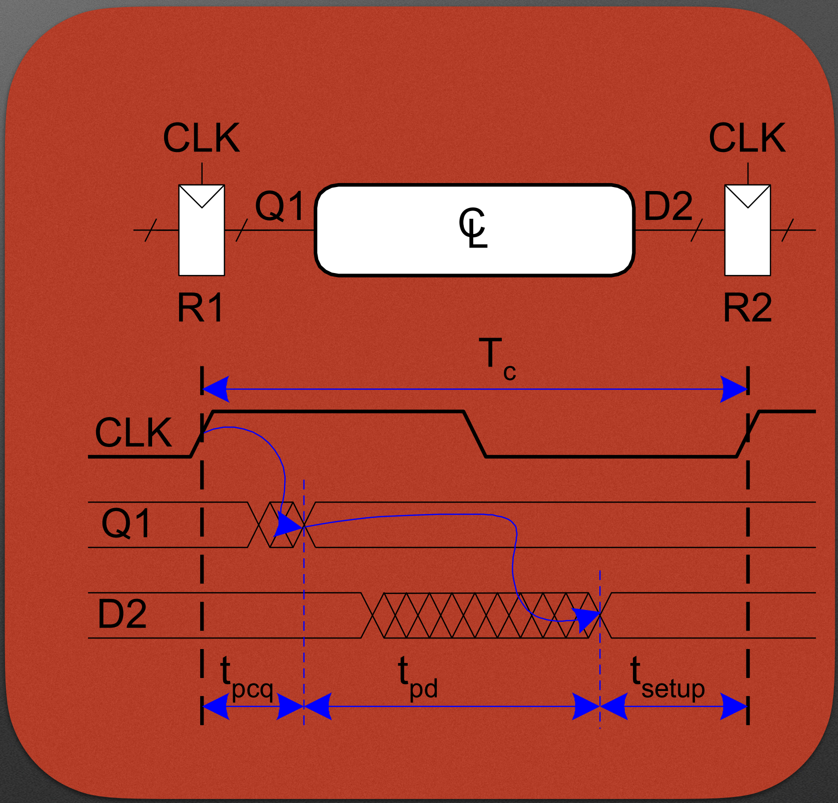
Setup Time Constraint

- Max time from R1 through CL
  - R2's input needs to be stable t_{setup} before clock



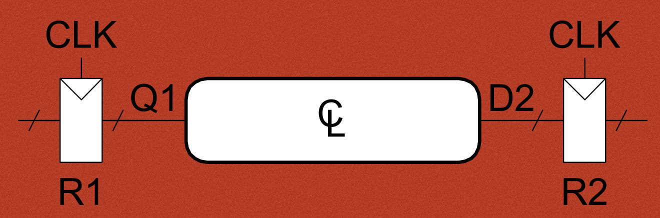
Setup Time Constraint

- Max time from R1 through CL
  - R2's input needs to be stable t_{setup} before clock



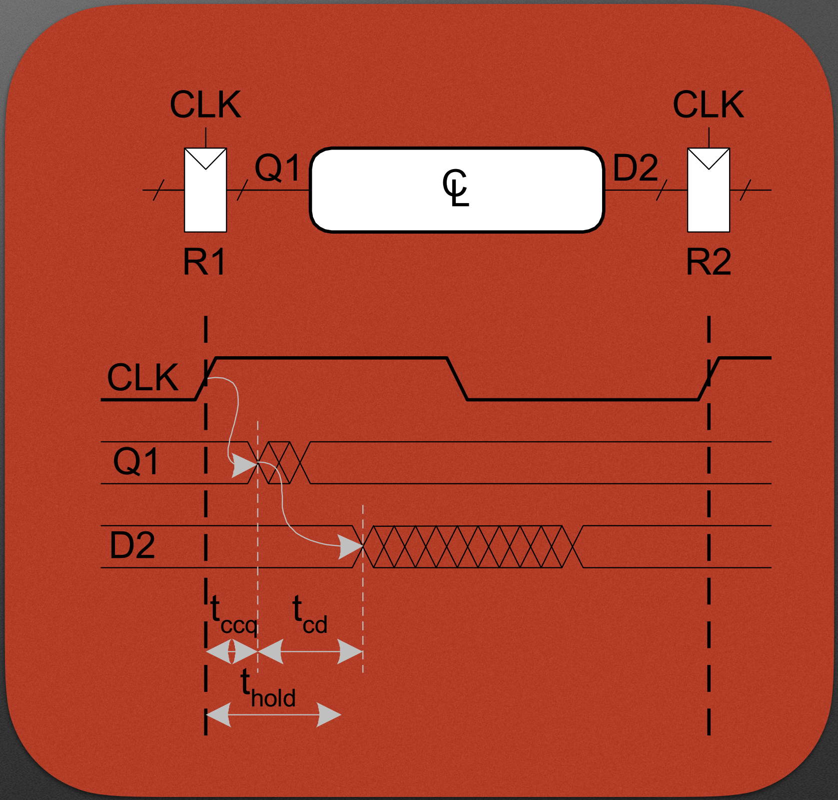
Hold Time Constraint

- Min time from R1 through CL
  - R2's input must be stable t_{hold} after the clock



Hold Time Constraint

- Min time from R1 through CL
  - R2's input must be stable t_{hold} after the clock



Synchronous Timing

- Must meet both
  - Setup Time Constraint
  - Hold Time Constraint

Chapter 4

Hardware Description Languages (HDLs)

- Specifies logic function only
- Computer-aided design (CAD) tool produces or synthesizes the optimized gates
- Most commercial designs built using HDLs

VHDL

- VHDL 2008
- Developed in 1981 by the Department of Defense
- IEEE standard (1076) in 1987
- Updated in 2008 (IEEE STD 1076-2008)

(System) Verilog

- Developed in 1984 by Gateway Design Automation
- IEEE standard (1364) in 1995
- Extended in 2005 (IEEE STD 1800-2009)

HDL

- A HDL is not a computer program.
- A HDL is not a computer program!
- A HDL is *not* a computer program!
- A HDL is <u>*not*</u> a computer program!
- A HDL is <u>***NOT***</u> a computer program!

HDL

- HDL is a way to *describe* hardware
- HDLs typically can describe hardware in different ways
- HDLs describe hardware in modules

(System) Verilog Module Example

(System) Verilog Module Example


```
module example(input  logic a, b, c,
               output logic y);
  // module body goes here
endmodule
```

(System) Verilog

- Case sensitive
- Identifiers follow rules like in programming languages
  - Ex: `2inputGate` not allowed, but `gateWithTwoInputs` is.
- Whitespace ignored
- C/Java Style comments:

```
// Comment to end of line
/*
  Block Comment
 */
```

(System) Verilog

- Basic logic operators
  - Binary: &, |, ^
  - Unary: ~

HDL: ***Structural*** (Verilog)

```
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                         // internal signal

  and3 andgate(a, b, c, n1);        // instance of and3
  inv  inverter(n1, y);             // instance of inv
endmodule
```

HDL: ***Structural*** (Verilog)

```
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                       // internal signal

  inv  inverter(n1, y);           // instance of inv
  and3 andgate(a, b, c, n1);      // instance of and3
endmodule
```

HDL: ***Behavioral*** (Verilog)

```
module nand3(input  logic a, b, c
             output logic y);
   assign y = ~(a & b & c);
endmodule
```

(System) Verilog

- Multi-bit values: Array-like notation with bit ordering:

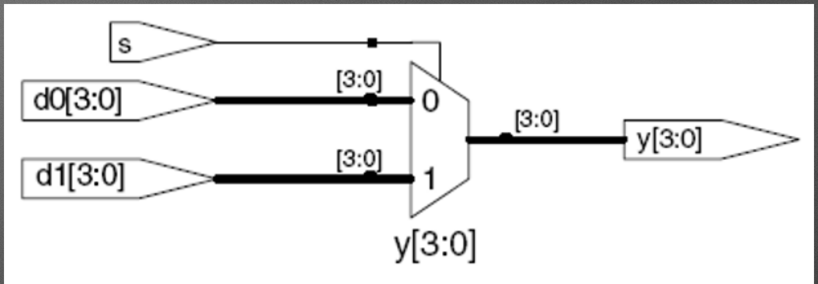
```
logic [7:0] a;
```

- Multi-bit reductions by pre-fix notation on multi-bit values

```
assign y = &a;
```

(System) Verilog

- Conditionals via Ternary operator (? :)




```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);
   assign y = s ? d1 : d0;
endmodule
```

Operator Precedence

<table>
<tr><td>~</td><td>NOT</td></tr>
<tr><td>*, /, %</td><td>mult, div, mod</td></tr>
<tr><td>+, -</td><td>add, sub</td></tr>
<tr><td><<, >></td><td>shift</td></tr>
<tr><td><<<, >>></td><td>arithmetic shift</td></tr>
<tr><td><, <=,</td><td>comparison</td></tr>
<tr><td>==, !=</td><td>equal, not equal</td></tr>
<tr><td>&, ~&</td><td>AND, NAND</td></tr>
<tr><td>^, ~^</td><td>XOR, XNOR</td></tr>
<tr><td>|, ~|</td><td>OR, NOR</td></tr>
<tr><td>?:</td><td>ternary</td></tr>
</table>

Number Formats

Number	# Bits	Base	Decimal	Stored
3'b101	3	binary	5	101
'b11	unsized	binary	3	00…0011
8'b11	8	binary	3	00000011
8'b1010_1011	8	binary	171	10101011
3'd6	3	decimal	6	110
6'o42	6	octal	34	100010
8'hAB	8	hexadecimal	171	10101011
42	Unsized	decimal	42	00…0101010

Verilog Bit Manipulations

- Subscripting and grouping

```
assign y = {a[2:1], {3{b[0]}}, a[0], 6'b100_010};
```

- Underscores can be used for clarity

Questions

- What about the exam????
- Will we use a HDL?
- [Which HDL?]
- How big will our circuits/computers get?
- What's synthesis? Are there things that can't be synthesized?
- Is my degree worth it? (I think mine was. Your mileage may vary…)

Next Time

- Studio
- Homework 4 due next Wed night!