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

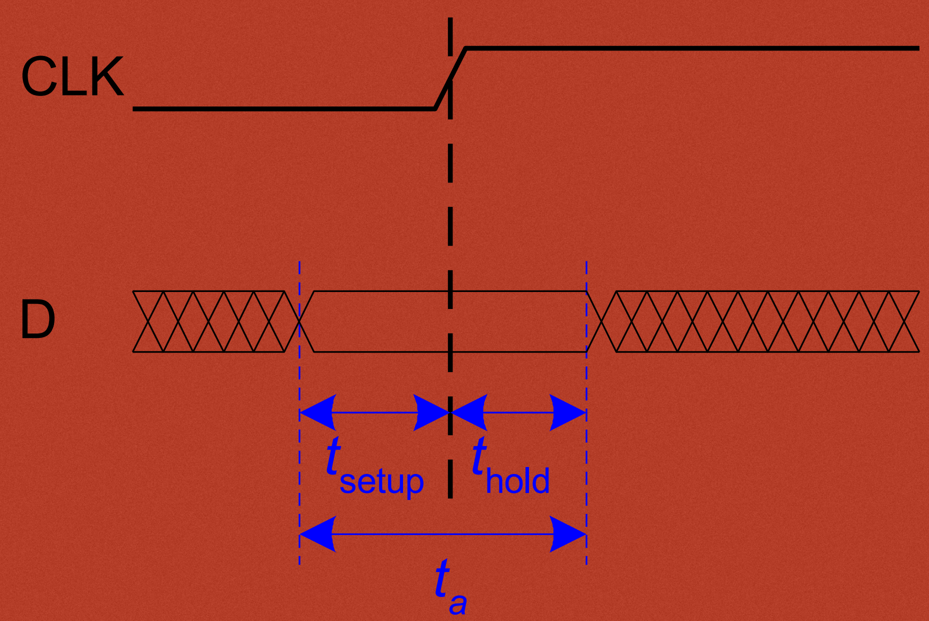
Bill Siever
&
Jim Feher

This week

- Thursday:
  Studio — Here / Seigle 301
- Midterm Grades: Canvas
  - Studios and Pre-lecture grades posted
  - Hw 1-4 returned / grades posted
  - Exam 1 returned / grades posted

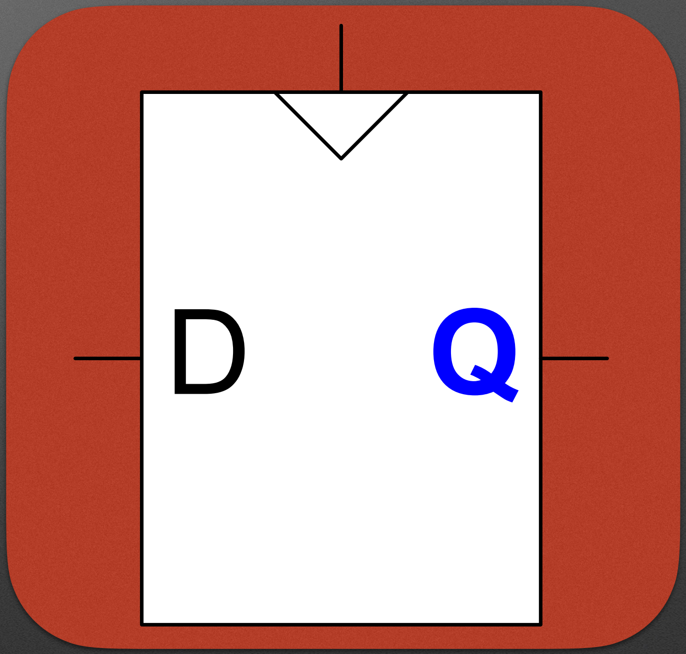
Exam

Dff: Setup & Hold Time

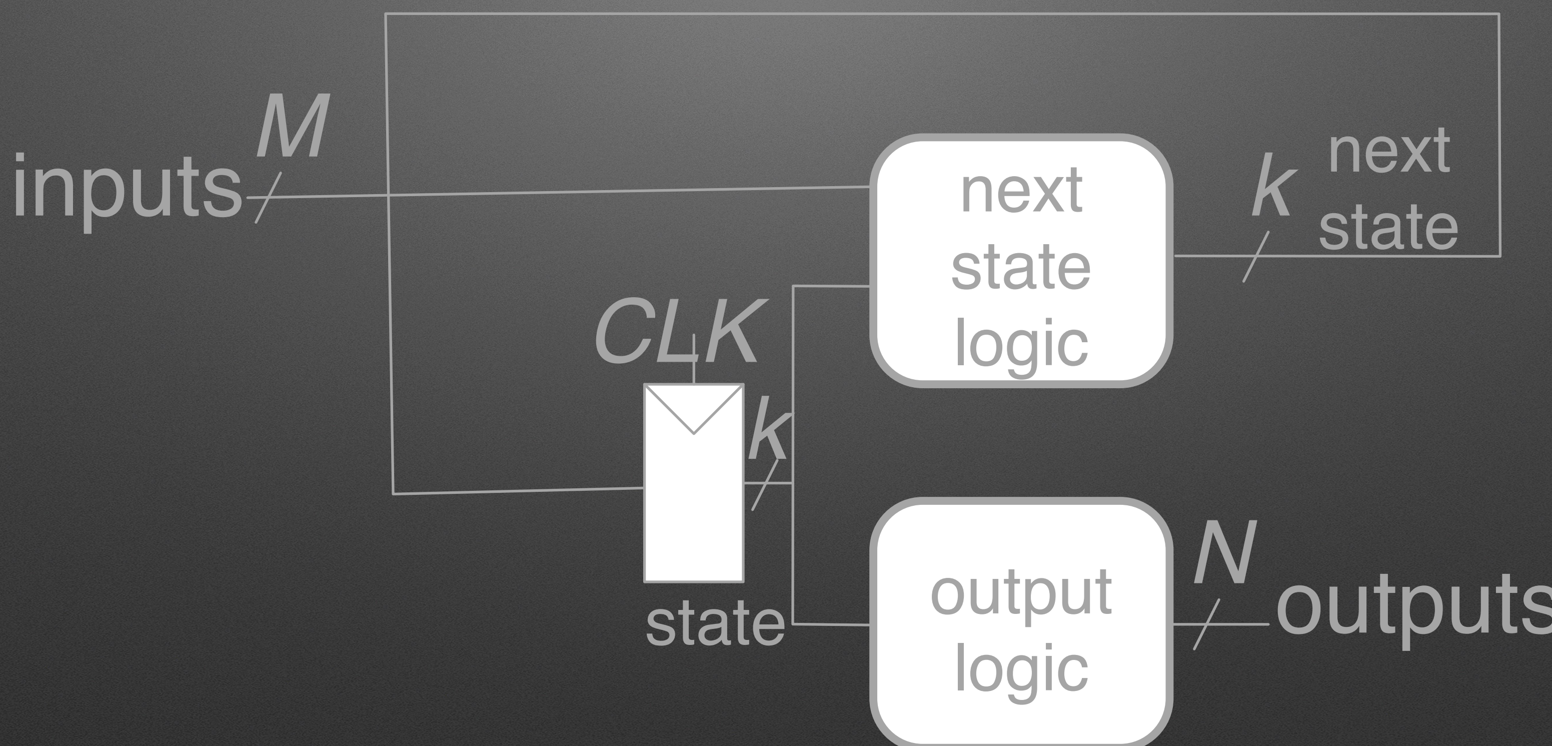


Dff Time Parameters

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

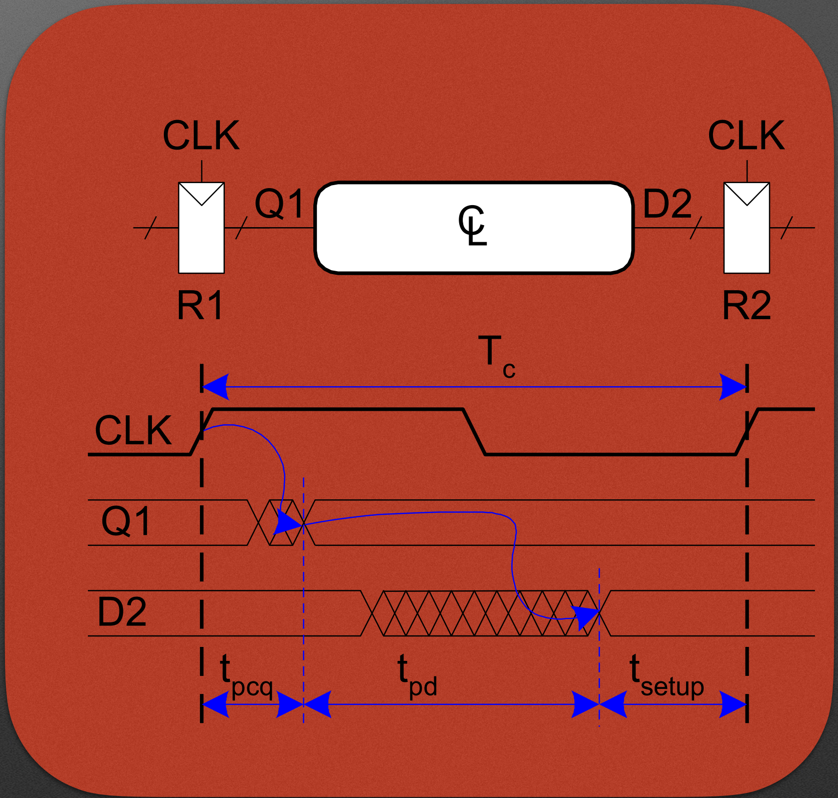


Moore Machine: How fast can it go?



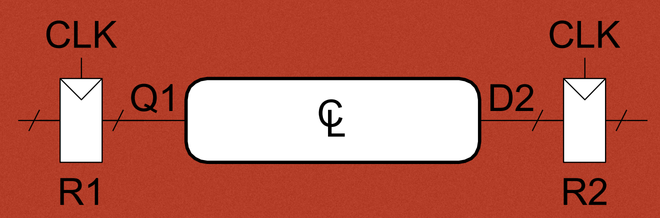
Fastest Clock Rate?

- R2's input needs to be stable t_{setup} before clock
- Variation of Moore Machine:
  Imagine no R2 - a loop to R1
- $T_c \geq t_{pcq} + t_{pd} + t_{setup}$



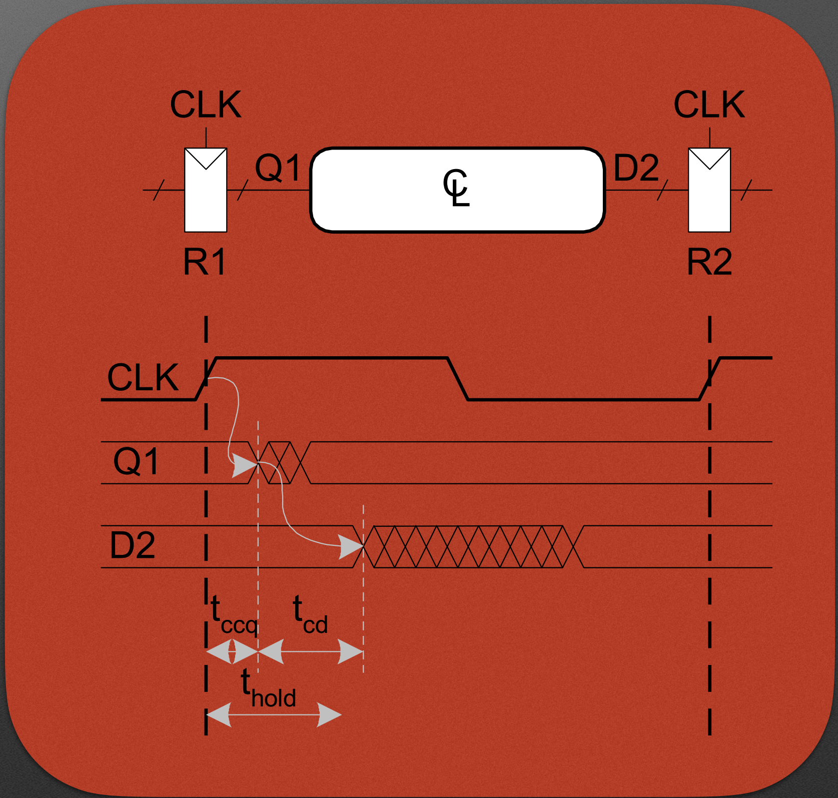
Hold Time Constraint

- 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
  - Both t_{ccq} and t_{hold} are provided/set
- $t_{cd} \geq t_{hold} - t_{ccq}$
  May need to add delays in Combo. Logic to meet



Chapter 4

HDL

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

HDLs *Describe* Hardware

- Uses
  - Simulation: Confirm modules work together
  - "Synthesis" : Transformation to real hardware
    - Like compilers used for programming languages
- Description Styles
  - Structure (connect 2 input AND to …)
  - Behavior (if x then y)

HDL

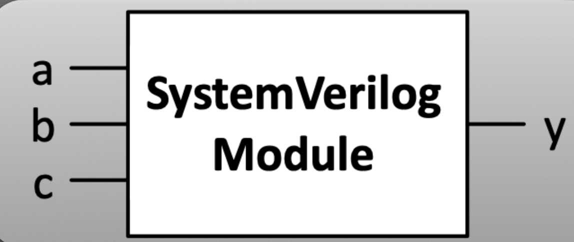
- A HDL is not a computer program.
- A HDL is not a computer program!
- A HDL is *not* a computer program!
- A HDL is <u>*not*</u> a computer program!
- A HDL is <u>***NOT***</u> a computer program!

We will use Verilog

Not VHDL

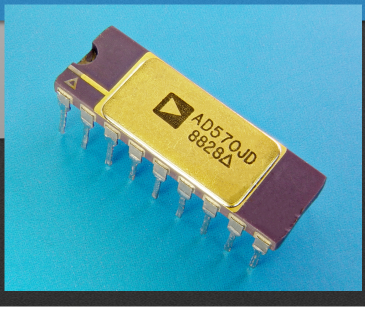
(System) Verilog Module Example

(System) Verilog Module Example

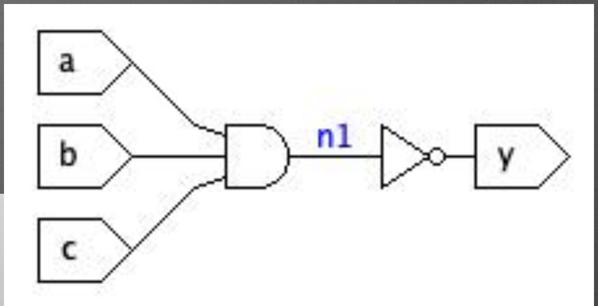



```
module example(input  logic a, b, c,
               output logic y);
  // module body goes here
endmodule
```

Input & Output are like the Pins On chips or in JLS



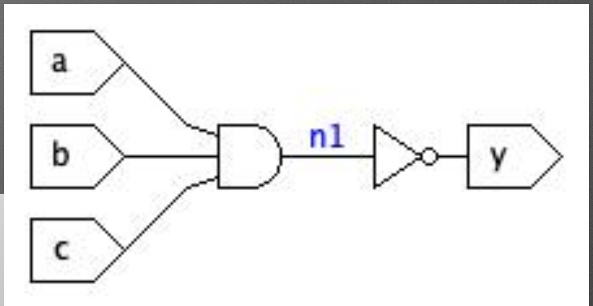
HDL: ***Structural*** (Verilog)




```
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)

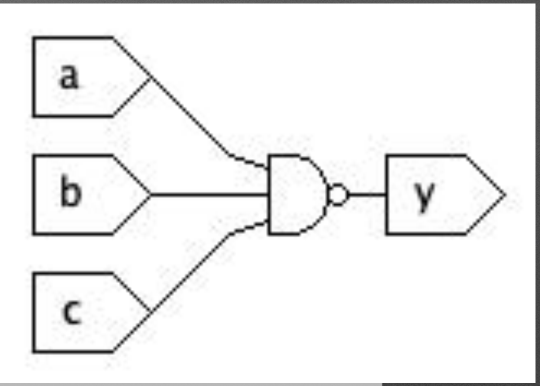



```
module nand3(input  logic a, b, c
             output logic y);
  logic n1;                          // internal signal

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

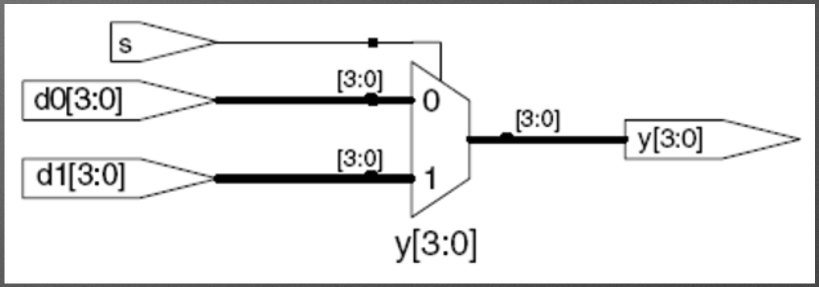
HDL: ***Behavioral*** (Verilog)

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



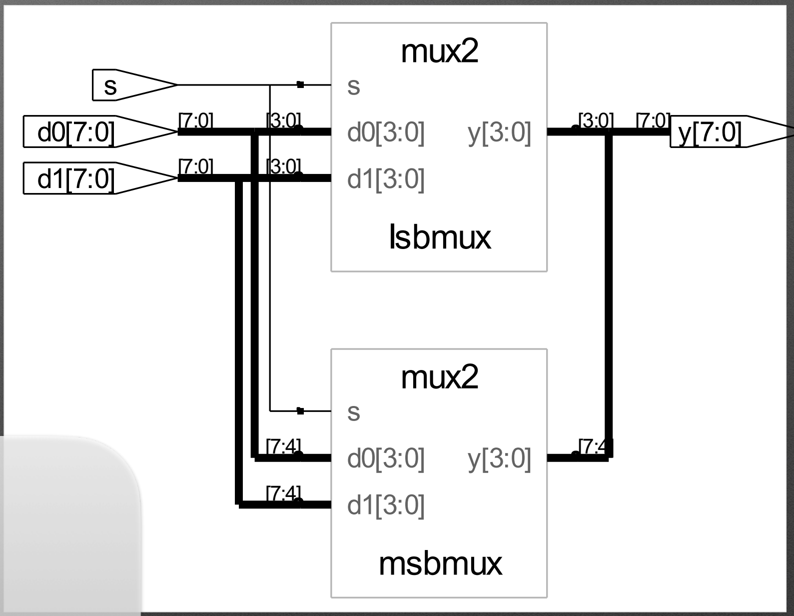
4-bit mux2: Behavior

- Conditionals via Ternary operator (? :)
- Multi-bit values via [] notation




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

8-bit mux2: Hierarchical




```
module mux2_8(input  logic [7:0] d0, d1,
              input  logic       s,
              output logic [7:0] y);

  mux2 lsbmux(d0[3:0], d1[3:0], s, y[3:0]);
  mux2 msbmux(d0[7:4], d1[7:4], s, y[7:4]);
endmodule
```

Sequential Logic

`always`: Based on *Events*

- Concept of “event” is related to simulation and “event driven programming”
- JLS uses events: An OR gate “reacts” to events and schedules an update
  See here

`always` Statement

- Form:

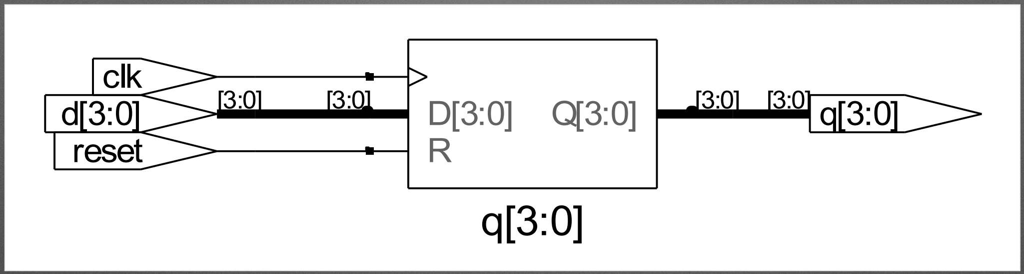
```
always @(sensitivity list)
    statement;
```

- When event in sensitivity list occurs, statement is executed

Verilog: D Flip-Flop


```
module flop(input  logic       clk,
            input  logic [3:0] d,
            output logic [3:0] q);
  always_ff @(posedge clk)
    q <= d;            // pronounced “q gets d”
endmodule
```

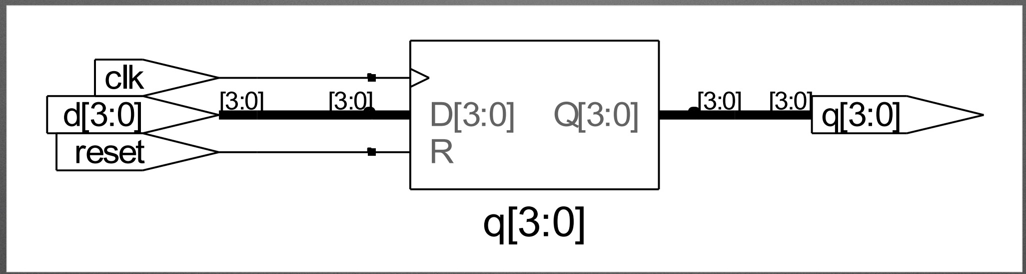
Resettable D-Flip-Flop 1




```
module flopr(input  logic       clk,
             input  logic       reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_ff @(posedge clk)
    if (reset) q <= 4'b0;
    else       q <= d;
endmodule
```

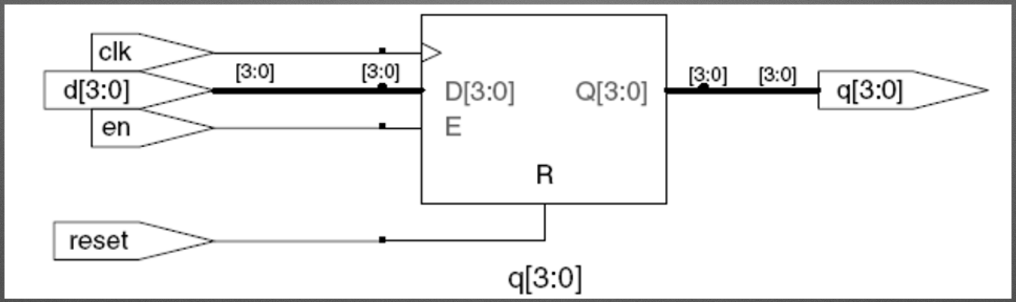
Resettable D-Flip-Flop 2




```
module flopr(input  logic       clk,
             input  logic       reset,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_ff @(posedge clk, posedge reset)
    if (reset) q <= 4'b0;
    else       q <= d;
endmodule
```

Resettable D-Flip-Flop 3




```
module flopr(input  logic       clk,
             input  logic       reset,
             input  logic       en,
             input  logic [3:0] d,
             output logic [3:0] q);

  always_ff @(posedge clk, posedge reset)
    if (reset)    q <= 4'b0;
    else if (en)  q <= d;
endmodule
```

`always` and Combinational Logic

```
always_comb
   begin
      y = a & b
      …
   end
```

Block of assignments

Could have been done with individual `assign`s

Notice = (“blocking assignment”), not <= (“non-blocking assignment”)

`always_comb` has nice features

- `case` : Selection between several options
  Great for state machines!

- Must describe all possible combinations to be comb logic. Use `default`

```
case (state)
  soap:                hot = 1;
  highPressureWarm: hot = 1;
  …
  default: hot = 0;
 endcase
```

Questions

- Will we use a HDL? Yes
- [Which HDL?]: Verilog
- Test benches?
- Is my degree worth it? (I think mine was. Your mileage may vary…)

Next Time

- Studio