Intro to Digital Design
L8: Project Tips, Memory

Instructor: Naomi Alterman

Teaching Assistants:

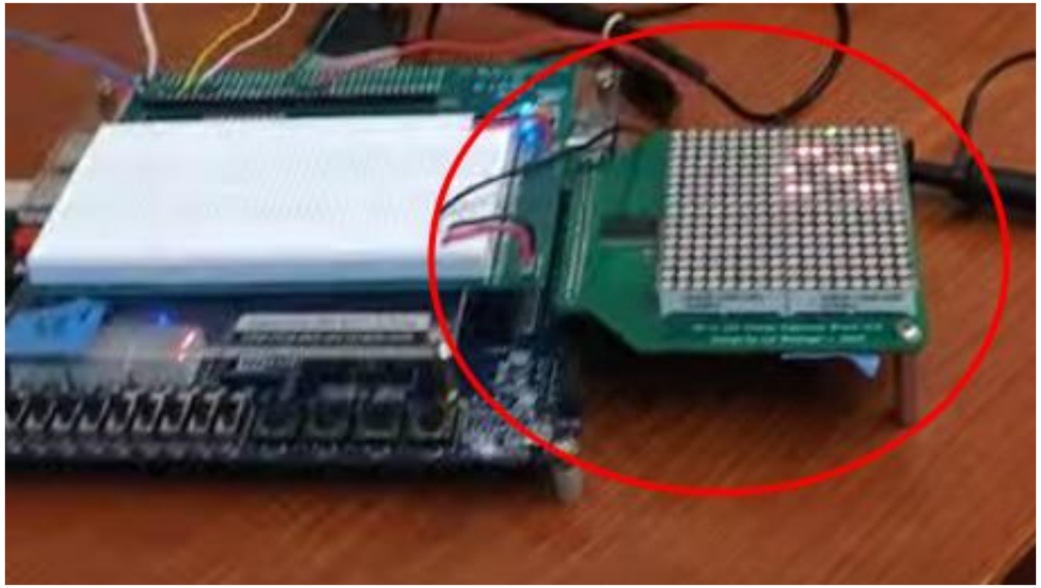
Derek de Leuw
Kevin Hernandez
Aadithya Manoj
Isabel Froelich
Sathvik Kanuri

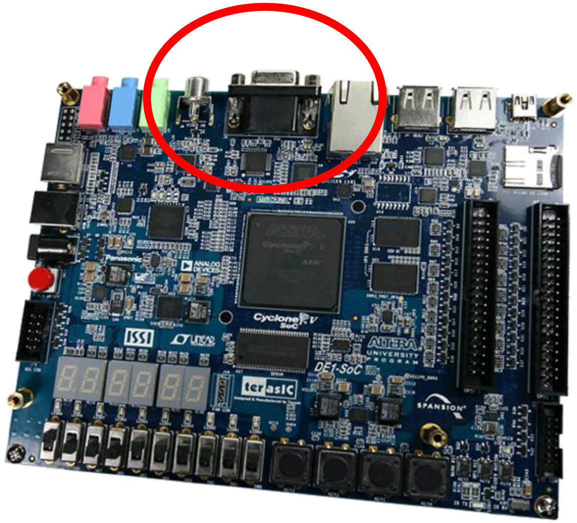
Relevant Course Information

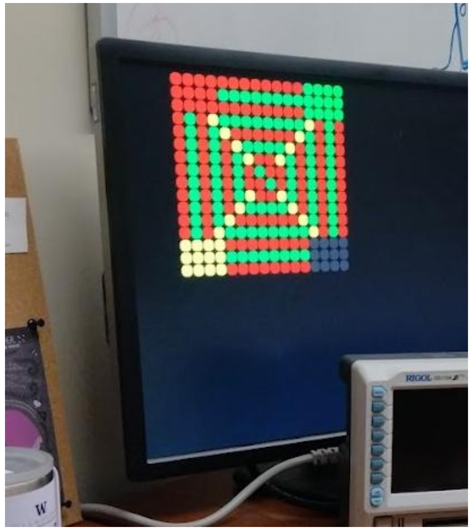
- Lab 8 – Project! / “Big Homework”
  - 2 weeks to work on it – **don’t wait to start**!
    - Reports due Friday, March 13 @ 11:59 pm
    - Lab 8 check-in due next week during demo slot
      - Submission **required!** Includes a block diagram of your overall system and at least one implemented module
    - Demos can be scheduled outside of the lab hours by making a *private* post on Ed Discussion
  - 8 suggested projects, or get your own approved
    - Not all are worth the same number of points (“full credit” is 150)
    - Think carefully about what you want to tackle (*e.g.*, complex FSM, LED board, multiple “clock speeds”)
  - Bonus points for adding cool features and early finish
    - Up to 20 points for extra features; up to 10 points for early finish

LED Breakout Boards

- Most final projects involve these LED breakout boards
- This quarter, some kits don't have them 😔
- For those that don't: we provide a drop-in "adapter" module to output the image to a VGA monitor
  - There are several such monitors in the lab
  - Plug into the top of your board







Practice

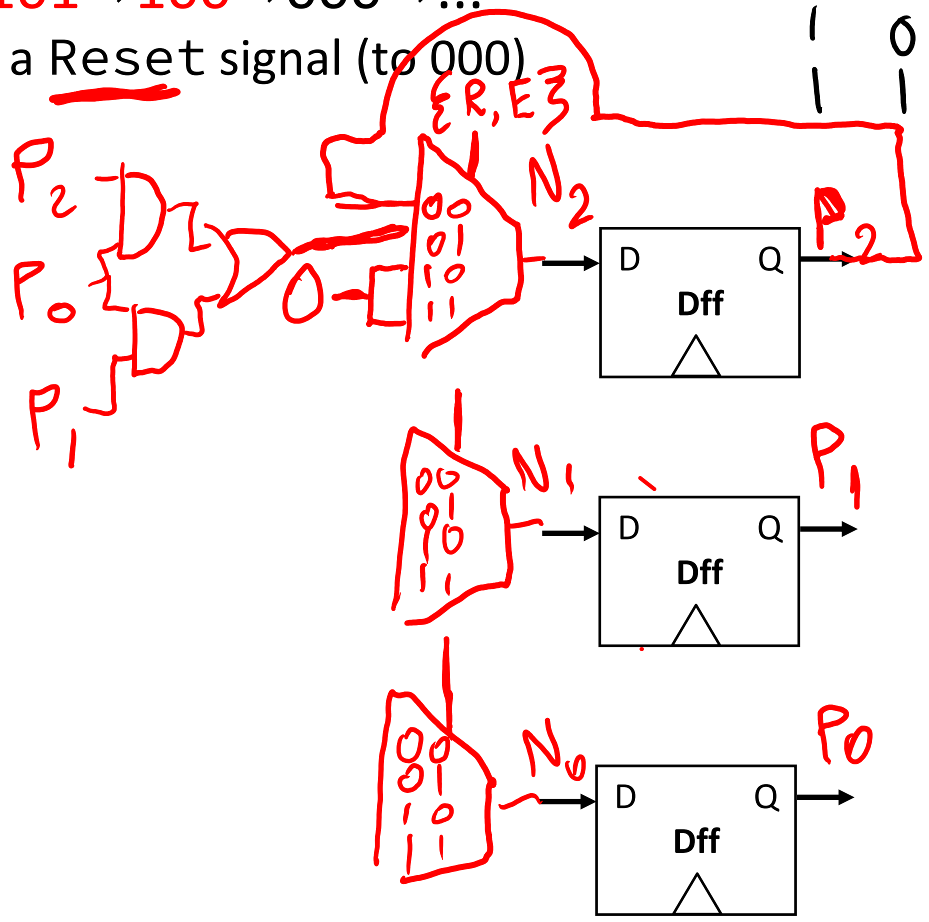
Action Table!

R	E	Action
0	0	N = P
0	1	N = Count Logic
1	0	N = 0
1	1	N = 0

- Implement a **counter** that goes through the state sequence 000→001→011→010→110→111→101→100→000→…
  - Include an Enable signal to count and a Reset signal (to 000)
  -

P_2	P_1	P_0	N_2	N_1	N_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

  - $N_2 = P_2 P_0 + P_1 \overline{P_0}$
  - $N_1 = \overline{P_2} P_0 + P_1 \overline{P_0}$
  - $N_0 = \overline{P_2}\,\overline{P_1} + P_2 P_1$



Outline

- **Project Tips**
  - **Comparators**
  - **Clock domains**
  - **Verilog `generate`**
  - **SystemVerilog Arrays**
- Memory
  - ROM

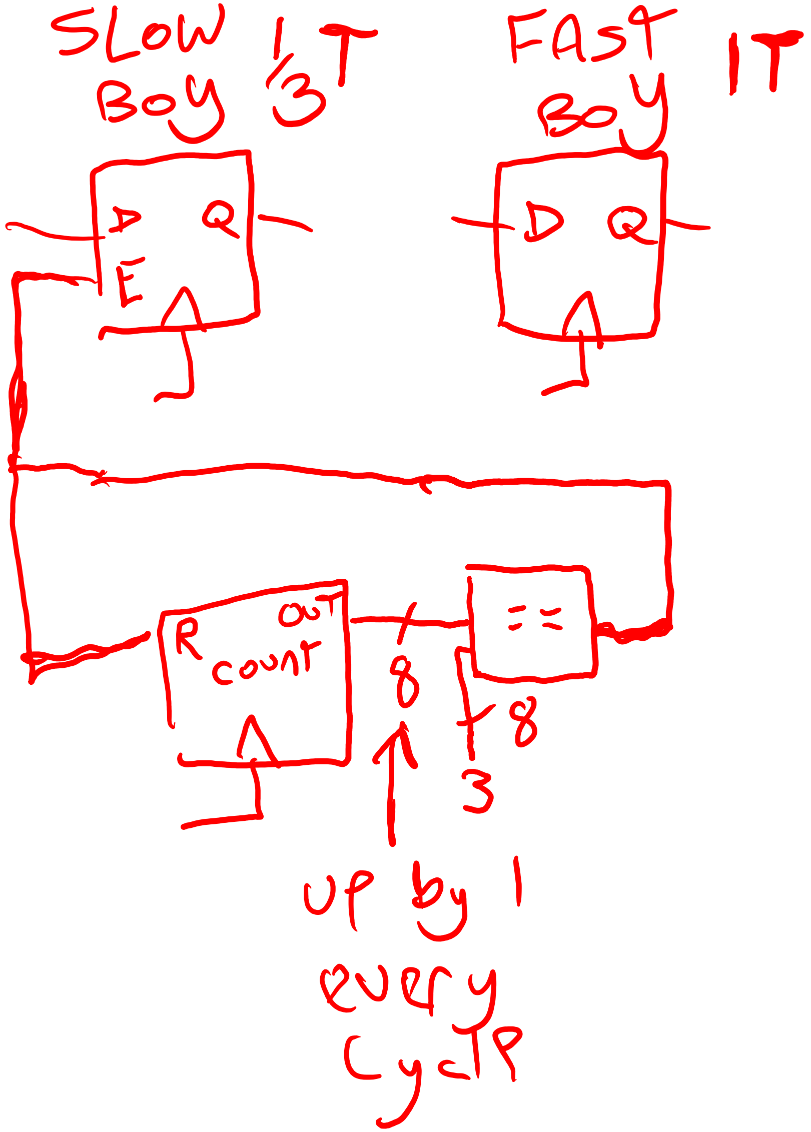
Comparators (Multibit)

A	B	NOR	XOR	XNOR
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

- Zero comparator A==0
  - NOR all of A’s bits
- Equality (A == B)
  - XNOR corresponding bits of A and B, then AND together
  - Compute A–B, then check if it == 0
- Comparator (A < B, A == B, A > B)
  - A < B: MSB of A–B
  - A == B: NOR of all bits of A–B
  - A > B: NOT of MSB of A–B

"Multiple Clocks" Via Enable Signals and Counters

- Even with that `clock_divider` module, we **only ever want to run our circuit on *one* clock if we can help it**
  - Logic stages that share a clock are called a **clock domain**
  - Crossing clock domains carelessly is a recipe for metastability
- So how can we slow things down?
  - Use flip flops with an Enable signal
  - Comparator on counter output generates an Enable pulse once every N cycles
  - Keeps all logic technically on the same clock domain, while allowing some things to happen "more slowly" than others



Advanced Verilog: `generate`

- Condense your code using loops and conditionals
  - Often used with `assign` and module instantiation

- Details:
  - Loop variables must be declared as `genvar` outside of `generate` statement
  - Block statements (`for`/`if`) *must* have `begin` and end and be labeled

```
genvar <loop_var>;

generate
  for (<init>; <cond>; <update>) begin : <label>
    // do something with loop_var
  end
endgenerate
```

Add/Sub in Verilog (parameterized)

- Variable-width add/sub (with overflow, carry)

```
module addN #(parameter N=32) (OF, CF, S, sub, A, B);
  output logic         OF, CF;
  output logic [N-1:0] S;
  input  logic         sub;
  input  logic [N-1:0] A, B;
  logic  [N-1:0] D;    // possibly flipped B
  logic          C2;   // second-to-last carry-out

  always_comb begin
    D = B ^ {N{sub}};  // replication operator
    {C2, S[N-2:0]} = A[N-2:0] + D[N-2:0] + sub;
    {CF, S[N-1]} = A[N-1] + D[N-1] + C2;
    OF = CF ^ C2;
  end
endmodule  // addN
```

Add/Sub in Verilog (generate)

- Generate produces N `fulladd` modules

```
module addNgen #(parameter N=32) (OF, CF, S, sub, A, B);
  output logic OF, CF;          // overflow and carry flags
  output logic [N-1:0] S;       // sum output bus
  input  logic sub;             // subtract signal
  input  logic [N-1:0] A, B;    // input busses
  logic [N:0]  C;               // carry signals between modules
                                                       tag
   genvar i;
  generate
     For (i=0; i<N; i=i+1) begin : adders
        fulladd fa (.Cout[i+1], .s(S[i])...
     end
   endgenerate
```

- Reminder: `module fulladd (cout, s, cin, a, b);`

SystemVerilog Arrays

a bus

- A *bus* is known as a *vector* or packed array of bits
  - *e.g.*, `logic [31:0] divided_clocks;`
  - Can only be made of single bit datatypes
- “Regular” array syntax is known as an unpacked array
  - *e.g.*, `logic an_unpacked_array[4:0];`
  - Can be made of any datatype
- Multidimensional arrays can be combinations of packed and unpacked dimensions
  - *e.g.*, `logic [3:0] two_D_array[4:0];`

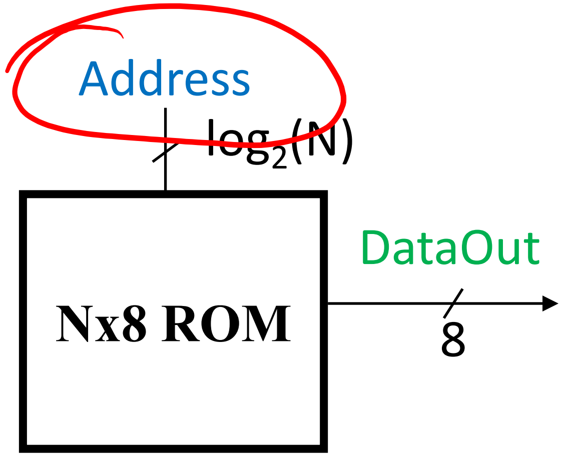
    “array of 5 4-bit busses”
  - Accessed left to right, starting with unpacked dimensions

Outline

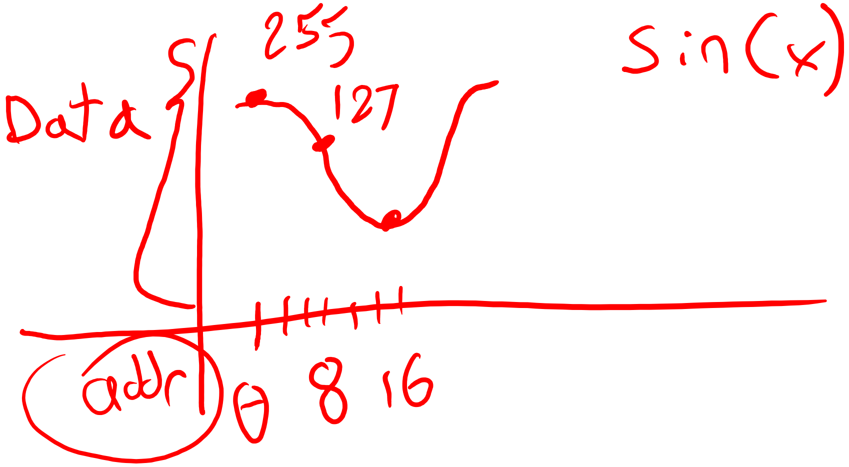
- Project Tips
  - Comparators
  - Clock domains
  - Verilog `generate`
  - SystemVerilog Arrays
- **Memory**
  - **ROM**

Storage Element: Idealized ROM

- “**R**ead **O**nly **M**emory”
  - NxM: An array of N M-Bit words
  - Address input selects the word driven on the output
  - Data “burned in” by the manufacturer (or your bitfile) and retained even if power is lost



- Difference between a ROM and your seven segment decoder is *density*
  - (and thus underlying circuit implementation)



- What can we do with a truly *read-only* memory?
  - Math look-up tables (trig functions)
  - FSM state transitions
  - Bitmap images and other simple patterns