Intro to Digital Design
Project Tips, Memory

Instructor: Justin Hsia

Teaching Assistants:

Emilio Alcantara
Eujean Lee
Naoto Uemura
Pedro Amarante
Wen Li

Relevant Course Information

- Lab 8 – Project
  - 2 weeks to work on it – don't wait to start!
    - Reports due Friday, May 31 @ 11:59 pm
    - Lab 8 check-in due next week during demo slot
    - 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
    - Most use LED breakout board – included in your lab kit
    - 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

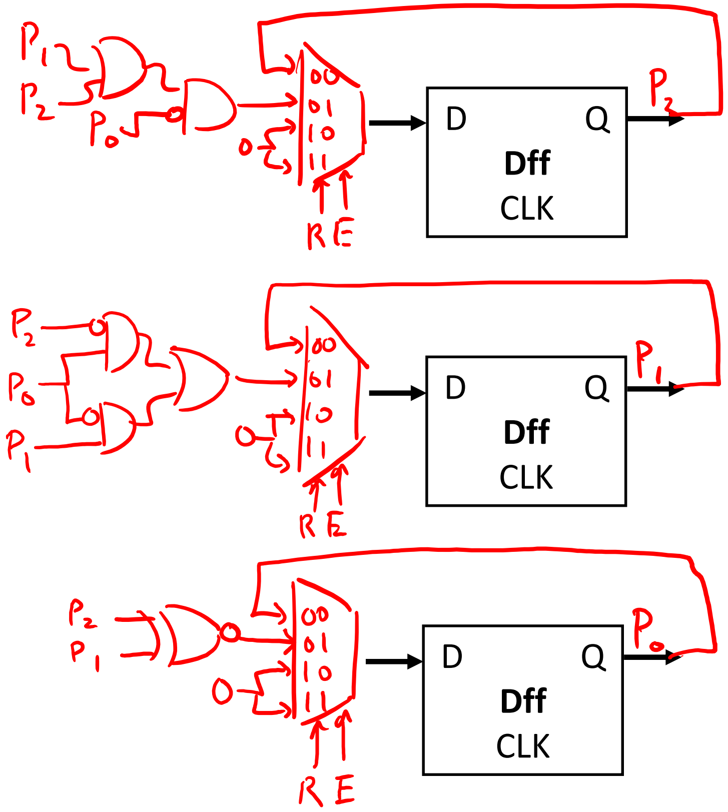
Practice

R	E	D
0	0	Q
0	1	N
1	0	0
1	1	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

  - $N_2 = P_2P_0 + P_1\overline{P_0}$
  - $N_1 = \overline{P_2}P_0 + P_1\overline{P_0}$
  - $N_0 = \overline{P_2}\,\overline{P_1} + P_2P_1$



Outline

- **Project Tips**
  - **“Multiple clocks”**
  - **Verilog `generate`**
  - **SystemVerilog Arrays**
- Computer Components
  - Memory/RAM

Comparator (Multibit)

- Equality (A == B)
  - XNOR corresponding bits of A and B, then AND together
  - NOR all bits of A–B

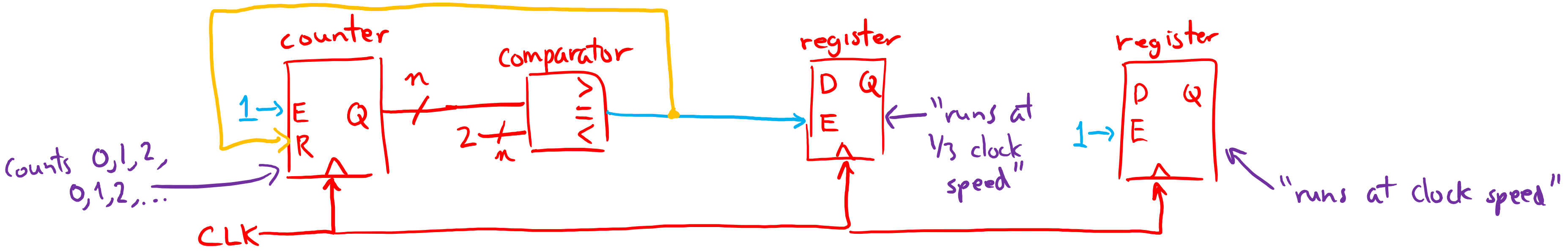
a	b	xnor	nor
0	0	1	1
0	1	0	0
1	0	0	0
1	1	1	0

- 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

computed just once!

"Multiple Clocks" Via Counters

- The `clock_divider` module is a 32-bit up counter
  - All output bits update at same time (t_{C2Q})
  - Output bits get us powers of 2 differences in speed
- Still want to use *single* clock for all state elements
  - We will instead control actions using the Enable signal
- Use comparator on a counter as Enable signal
  - May need to feedback into Reset signal on counter



Advanced Verilog: generate

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

testbenches:
integer i;
for(i=0; i<8; i=i+1) {
// set input signals
}

- 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
```

i
i=0 i<N i=i+1
adders
fullAdd fA ( );
use i here

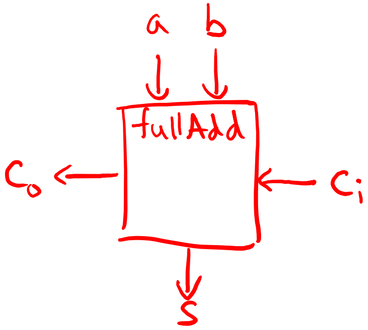
Add/Sub in Verilog (parameterized)

- 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

  genvar i;

  generate
    for (i=0; i < N; i=i+1) begin : adders
      fulladd fA (.cout(C[i+1]), .s(S[i]), .cin(C[i]), .a(A[i]), .b(B[i]));
    end
  endgenerate
```

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

SystemVerilog Arrays

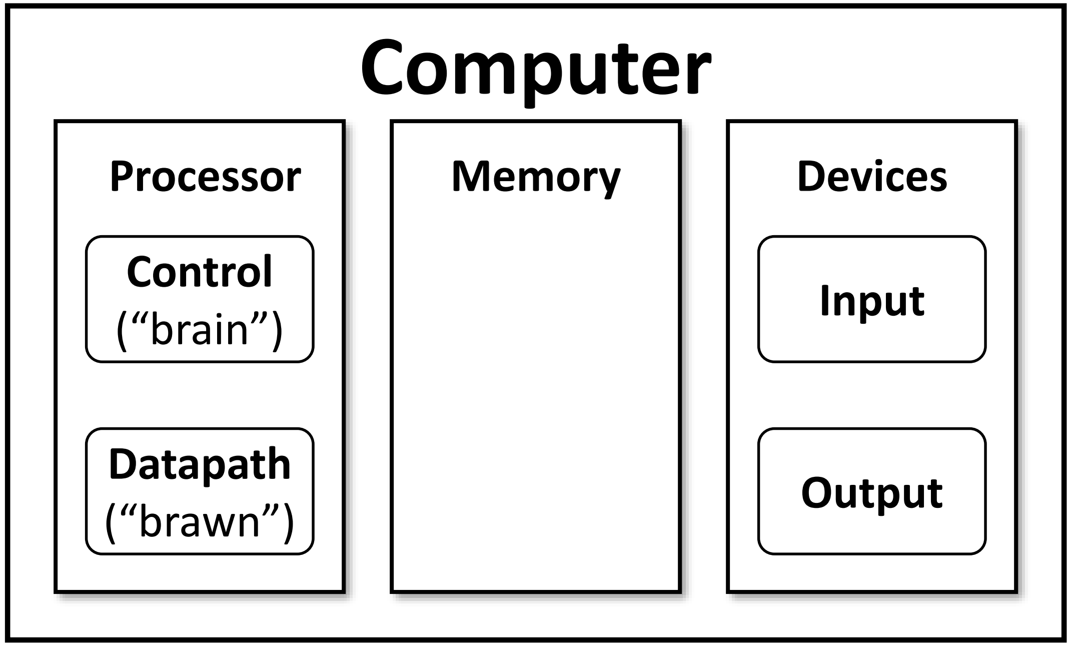
- A *bus* is known as a *vector* or packed array
  - *e.g.*, `logic [31:0] divided_clocks;`
  - Can only be made of single bit datatypes
- "Regular" array syntax is known as an unpacked array
  - *e.g.*, `logic an_unpacked_array[4:0];`
  - Can be made of any datatype
- Multidimensional arrays can be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

Outline

- Project Tips
  - "Multiple clocks"
  - Verilog `generate`
  - SystemVerilog Arrays
- **Computer Components**
  - **Memory/RAM**

Five Components of a Computer

- Components a computer needs to work:
  - Control
  - Datapath
  - Memory
  - Input
  - Output



Executing an Instruction

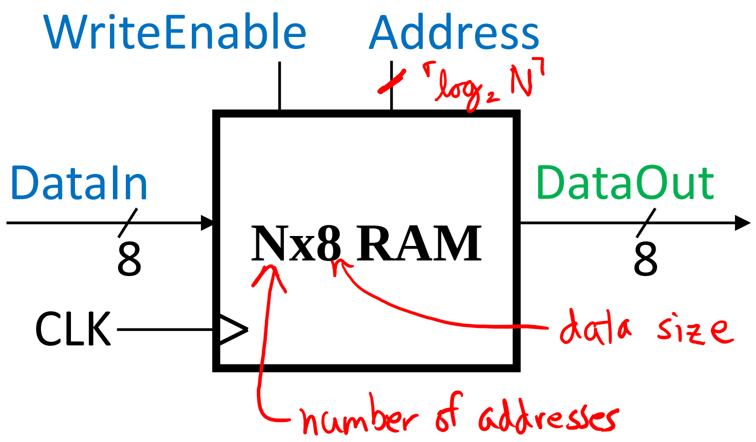
Example: addq (%rdi), %rax

Stack
Heap
Data
Code

- Depends on ISA, but generally:
  - Instruction Fetch ← get instruction from Memory (Code) (0x 48 03 07)
  - Instruction Decode ← what is this instruction telling us to do?
  - Data Fetch → register values, → memory
  - Computation → address computation, → instruction operation
  - Store Result → into register or memory
- Basic Datapath Components (idealized)
  - Register File } Next lecture
  - Memory Management Unit } Today
  - Arithmetic Logic Unit (ALU)
  - Routing Elements } Previous two lectures

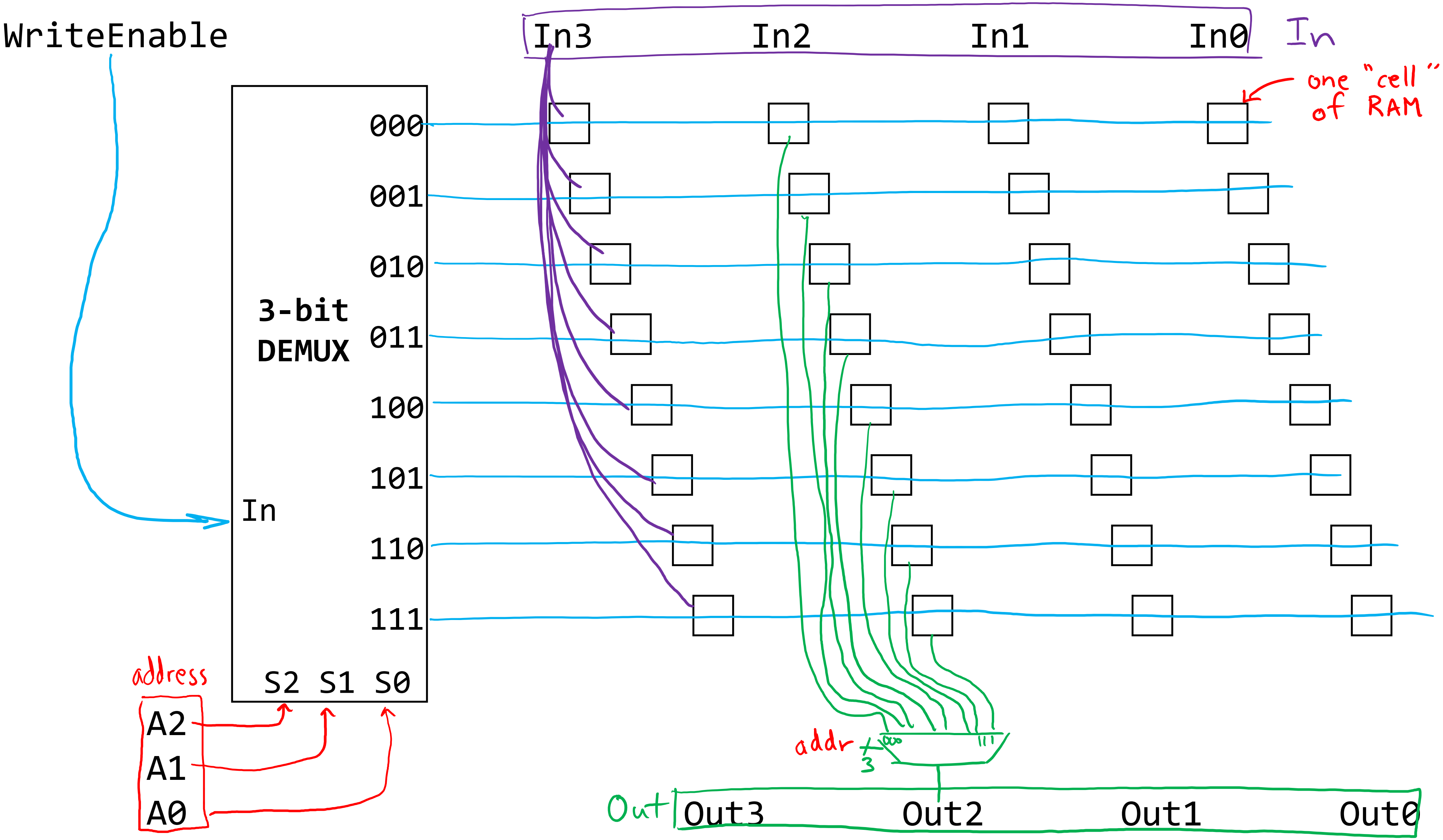
Storage Element: Idealized Memory

- Memory (idealized)
  - One input bus: DataIn
  - One output bus: DataOut
  - In reality, often combined



- Memory access:
  - Read: Data at Address placed on DataOut
  - Write: If WriteEnable = 1, DataIn written to Address
- For N addresses, need Address input to be $(\log_2 N)$-bits wide
- Clock (CLK) is a factor ONLY during write operation

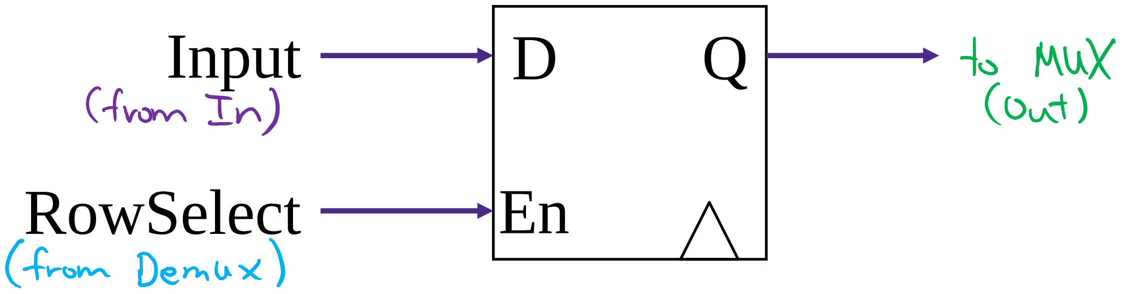
8x4 RAM


WriteEnable
In3
In2
In1
In0
In
one "cell" of RAM
3-bit DEMUX
In
000
001
010
011
100
101
110
111
S2 S1 S0
address
A2
A1
A0
addr
Out
Out3
Out2
Out1
Out0

RAM Cell

- Requirements:
  - Store one bit of data
  - Change data based on input when row is selected

- Just a controlled register!
  - No need to Reset
  - Use RowSelect as Enable



Verilog Memories

```
module memory16x8 (data_out, data_in, addr, write, clk);

  output logic [7:0] data_out;
  input  logic [7:0] data_in;
  input  logic [3:0] addr;
  input  logic       write, clk;

  logic        [7:0] mem [15:0];  // array of vectors

  assign data_out = mem[addr];

  always @(posedge clk)
    if (write)
      mem[addr] <= data_in;

endmodule  // memory16x8
```

both sides

first index accesses this dimension