Memory examples

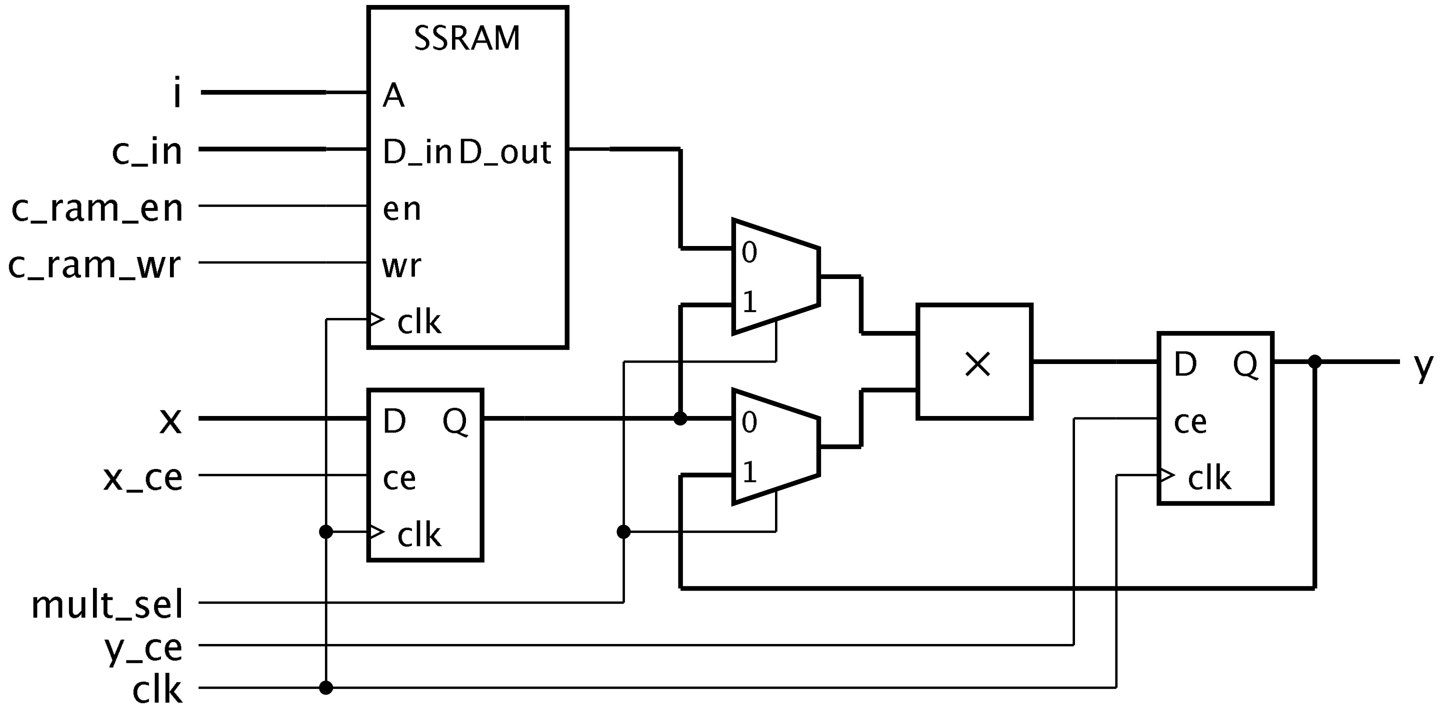
Chapter 5

Memories

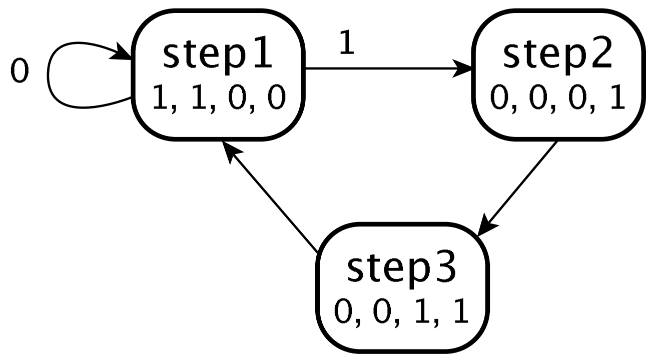
Example: Coefficient Multiplier

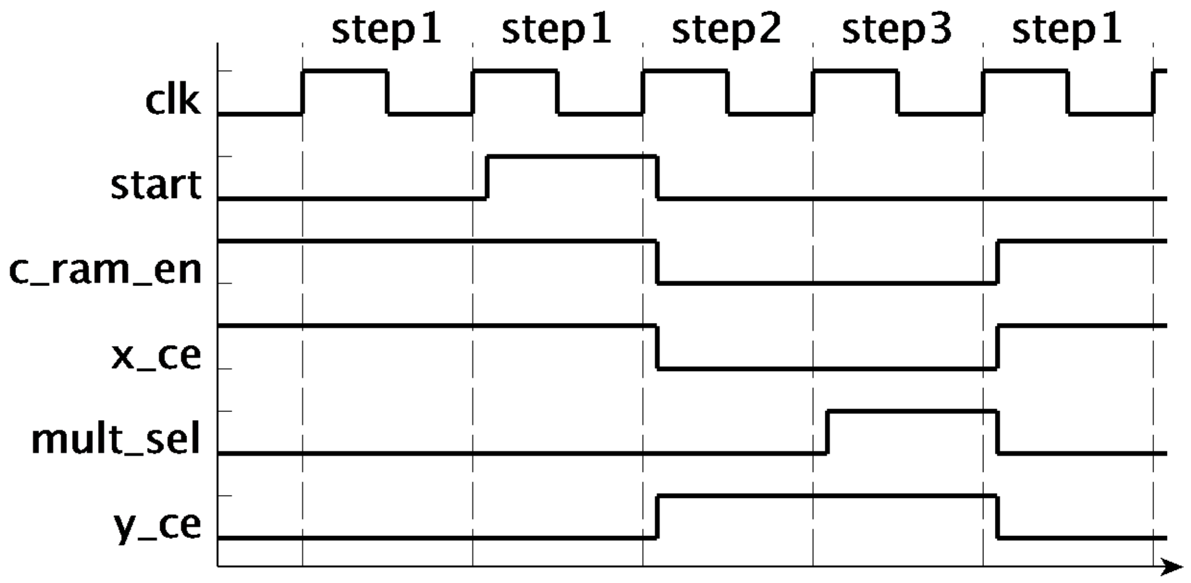
- Compute function $y = c_i \times x^2$
  - Coefficient stored in flow-through SSRAM
    - 12-bit unsigned integer index for i
  - x, y, c_i 20-bit signed fixed-point
    - 8 pre- and 12 post-binary point bits
  - Use a single multiplier
    - Multiply $c_i \times x \times x$

Multiplier Datapath



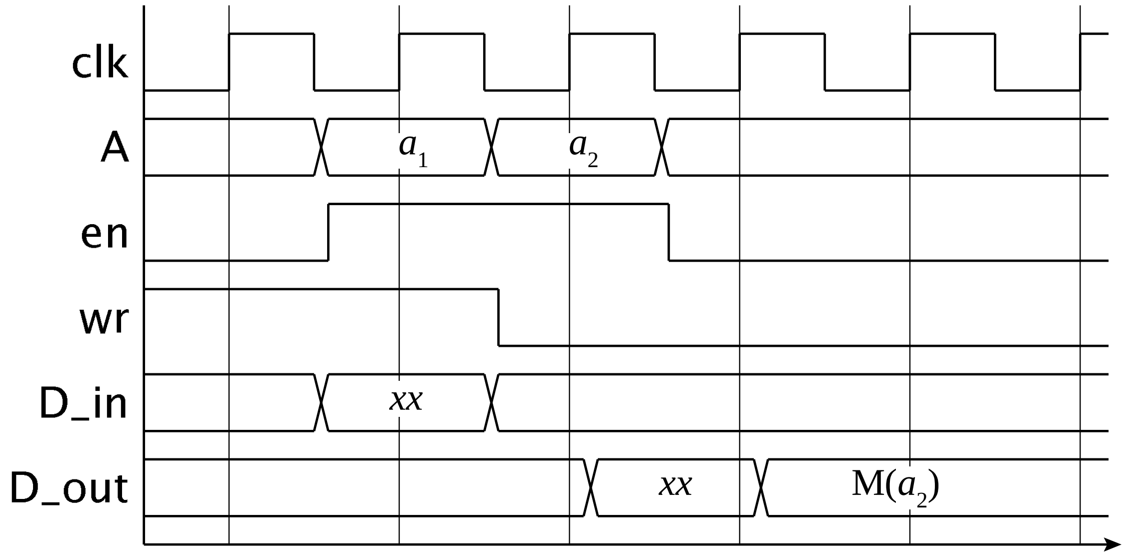
Multiplier Timing and Control





Pipelined SSRAM

- Data output also has a register
  - More suitable for high-speed systems
  - Access RAM in one cycle, use the data in the next cycle



Memories in Verilog

- RAM storage represented by an array variable

```
reg [15:0] data_RAM [0:4095];
...
always @(posedge clk)
  if (en)
    if (wr) begin
      data_RAM[a] <= d_in;  d_out <= d_in;
    end
    else
      d_out <= data_RAM[a];
```

Example: Coefficient Multiplier

```
module scaled_square ( output reg signed [7:-12] y,
                       input      signed [7:-12] c_in, x,
                       input             [11:0]  i,
                       input                     start,
                       input                     clk, reset );

  wire                   c_ram_wr;
  reg                    c_ram_en, x_ce, mult_sel, y_ce;
  reg signed [7:-12] c_out, x_out;

  reg signed [7:-12] c_RAM [0:4095];

  reg signed [7:-12] operand1, operand2;

  parameter [1:0] step1 = 2'b00, step2 = 2'b01, step3 = 2'b10;
  reg       [1:0] current_state, next_state;

  assign c_ram_wr = 1'b0;
```

Example: Coefficient Multiplier

```
always @(posedge clk)  // c RAM - flow through
  if (c_ram_en)
    if (c_ram_wr) begin
      c_RAM[i] <= c_in;
      c_out    <= c_in;
    end
    else
      c_out <= c_RAM[i];

always @(posedge clk)  // y register
  if (y_ce) begin
    if (!mult_sel) begin
      operand1 = c_out;
      operand2 = x_out;
    end
    else begin
      operand1 = x_out;
      operand2 = y;
    end
    y <= operand1 * operand2;
  end
```

Example: Coefficient Multiplier

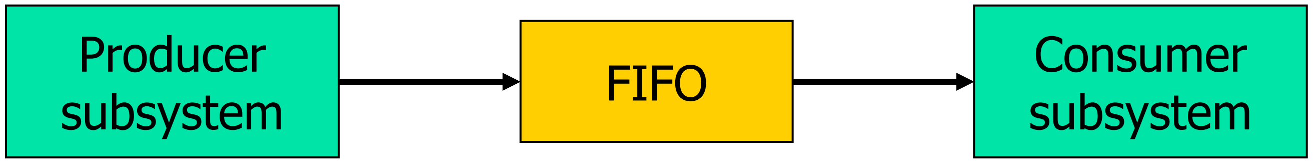
```
  always @(posedge clk)  // State register
    ...
  always @*  // Next-state logic
    ...
  always @* begin  // Output logic
    ...
endmodule
```

Multiport Memories

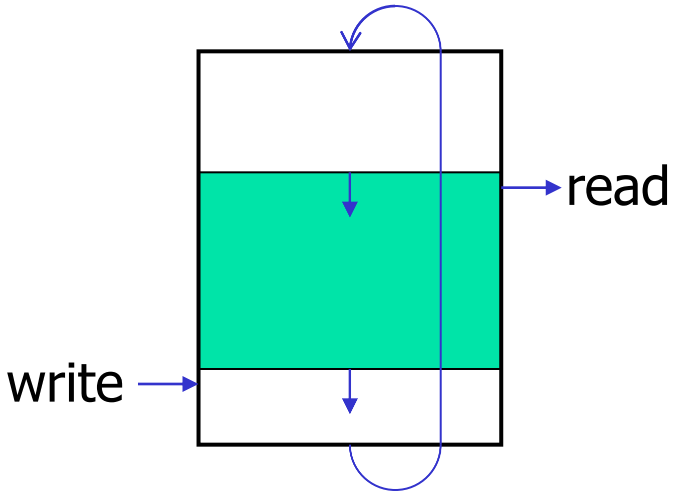
- Multiple address, data and control connections to the storage locations
- Allows concurrent accesses
  - Avoids multiplexing and sequencing
- Scenario
  - Data producer and data consumer
- What if two writes to a location occur concurrently?
  - Result may be unpredictable
  - Some multi-port memories include an arbiter

FIFO Memories

- First-In/First-Out buffer
  - Connecting producer and consumer
  - Decouples rates of production/consumption



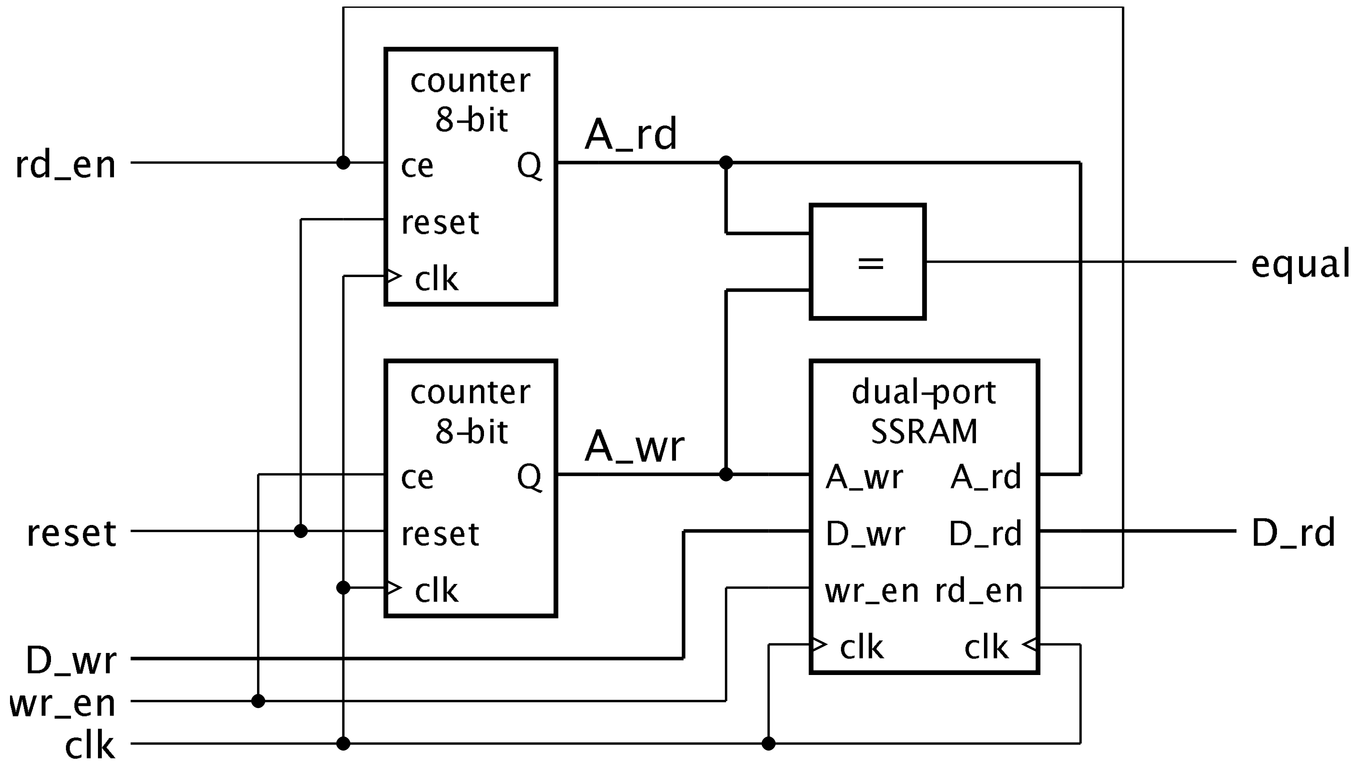
- Implementation using dual-port RAM
  - Circular buffer
  - Full: write-addr = read-addr
  - Empty: write-addr = read-addr



FIFO Example

- Design a FIFO to store up to 256 data items of 16-bits each, using 256x 16-bit dual-port SSRAM for the data storage. Assume the FIFO will not be read when it is empty, not to be written when it is full, and that the write and read ports share a common clock.

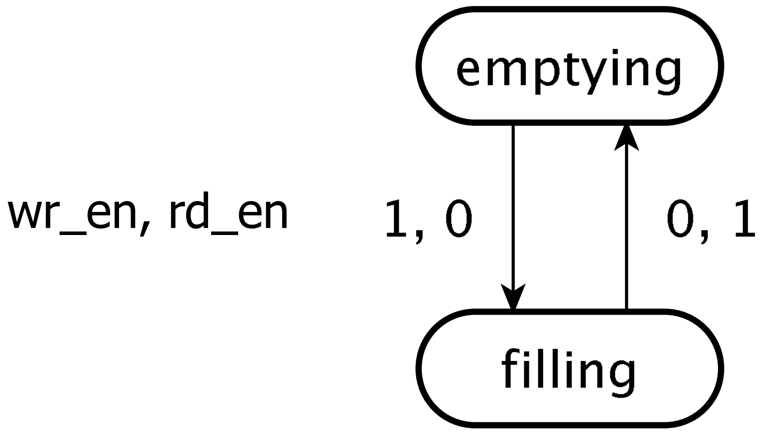
Example: FIFO Datapath



- Equal = full or empty
  - Need to distinguish between these states — How?

Example: FIFO Control

- Control FSM
  - → filling when write without concurrent read
  - → emptying when read without concurrent write
  - Unchanged when concurrent write and read



full = filling and equal

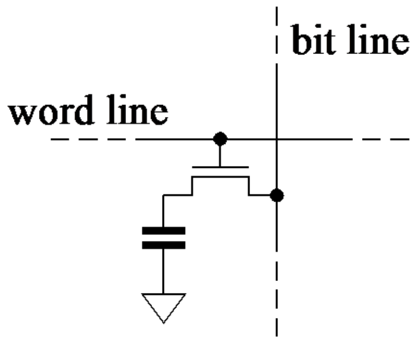
empty = emptying and equal

Multiple Clock Domains

- Need to resynchronize data that traverses clock domains
  - Use resynchronizing registers
- May overrun if sender's clock is faster than receiver's clock
- FIFO smooths out differences in data flow rates
  - Latch cells inside FIFO RAM written with sender's clock, read with receiver's clock

Dynamic RAM (DRAM)

- Data stored in a 1-transistor/1-capacitor cell
  - Smaller cell than SRAM, so more per chip
  - But longer access time
- Write operation
  - pull bit-line high or low (0 or 1)
  - activate word line
- Read operation
  - precharge bit-line to intermediate voltage
  - activate word line, and sense charge equalization
  - rewrite to restore charge



DRAM Refresh

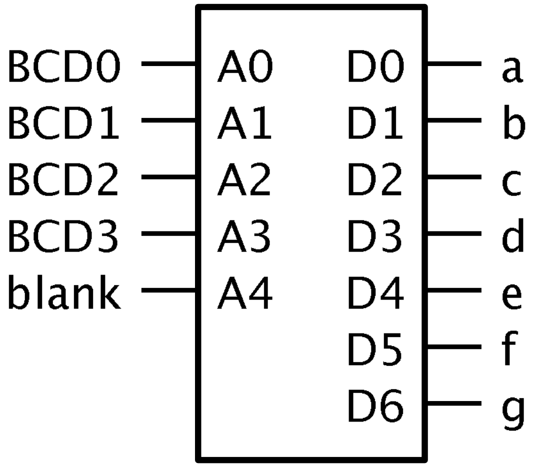
- Charge on capacitor decays over time
  - Need to sense and rewrite periodically
    - Typically every cell every 64ms
  - *Refresh* each location
- DRAMs organized into banks of rows
  - Refresh whole row at a time
- Can't access while refreshing
  - Interleave refresh among accesses
  - Or burst refresh every 64ms

Read-Only Memory (ROM)

- For constant data, or CPU programs
- Masked ROM
  - Data manufactured into the ROM
- Programmable ROM (PROM)
  - Use a PROM programmer
- Erasable PROM (EPROM)
  - UV erasable
  - Electrically erasable (EEPROM)
  - Flash RAM

Combinational ROM

- A ROM maps address input to data output
  - This is a combinational function!
  - Specify using a table
- Example: 7-segment decoder



Address	Content	Address	Content
0	0111111	6	1111101
1	0000110	7	0000111
2	1011011	8	1111111
3	1001111	9	1101111
4	1100110	10–15	1000000
5	1101101	16–31	0000000

Example: ROM in Verilog

```
module seven_seg_decoder ( output reg [7:1] seg,
                           input      [3:0] bcd,
                           input            blank );

  always @*
    case ({blank, bcd})
      5'b00000: seg = 7'b0111111;   // 0
      5'b00001: seg = 7'b0000110;   // 1
      5'b00010: seg = 7'b1011011;   // 2
      5'b00011: seg = 7'b1001111;   // 3
      5'b00100: seg = 7'b1100110;   // 4
      5'b00101: seg = 7'b1101101;   // 5
      5'b00110: seg = 7'b1111101;   // 6
      5'b00111: seg = 7'b0000111;   // 7
      5'b01000: seg = 7'b1111111;   // 8
      5'b01001: seg = 7'b1101111;   // 9
      5'b01010, 5'b01011, 5'b01100,
      5'b01101, 5'b01110, 5'b01111:
                seg = 7'b1000000;   // "-" for invalid code
      default:  seg = 7'b0000000;   // blank
    endcase

endmodule
```

Flash RAM

- Non-volatile, readable (relatively fast), writable (relatively slow)
- Storage partitioned into blocks
  - Erase a whole block at a time, then write/read
  - Once a location is written, can't rewrite until erased
- NOR Flash
  - Can write and read individual locations
  - Used for program storage, random-access data
- NAND Flash
  - Denser, but can only write and read block at a time
  - Used for bulk data, e.g., cameras, memory sticks

Summary

- Memory: addressable storage locations
- Read and Write operations
- Asynchronous RAM
- Synchronous RAM (SSRAM)
- Dynamic RAM (DRAM)
- Read-Only Memory (ROM) and Flash
- Multiport RAM and FIFOs