Department of Electronics & Communication Engg.

Course : Digital System Design using Verilog. **Sem.: 6th (2017-18)**

Course Coordinator:

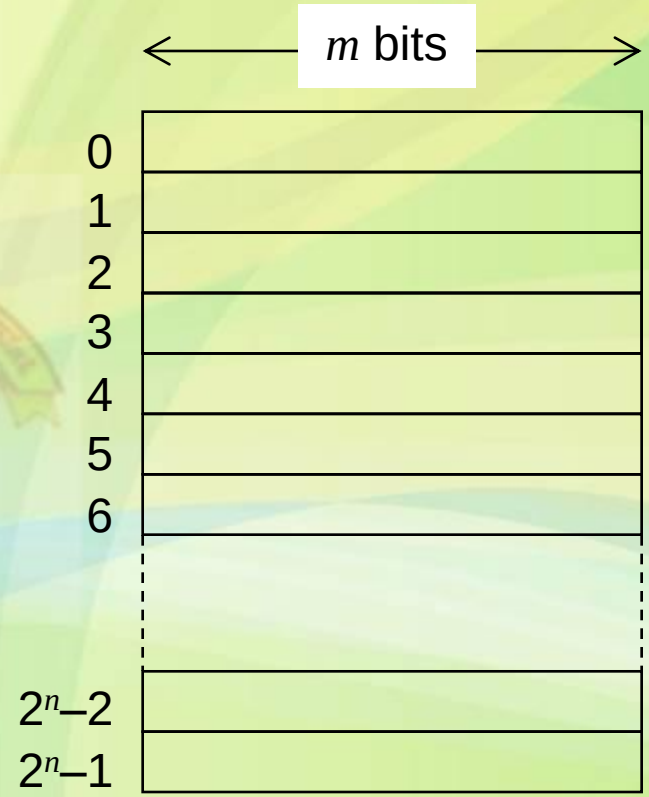
Prof. D. M. Kumbhar

Digital System Design Using Verilog

Module 2
Memories

General Concepts

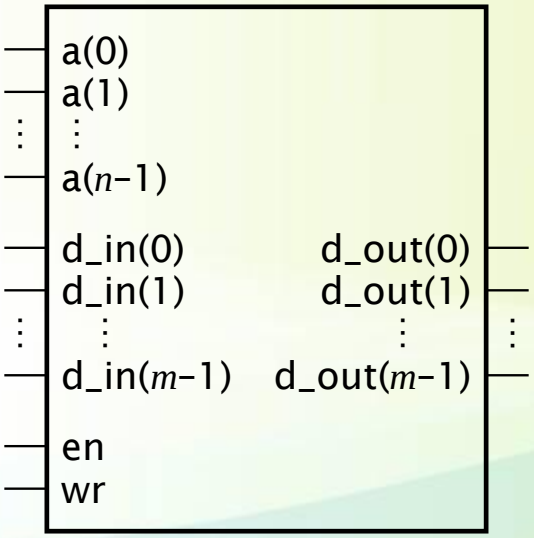
- A memory is an array of storage locations
  - Each with a unique address
  - Like a collection of registers, but with optimized implementation
- Address is unsigned-binary encoded
  - n address bits $\Rightarrow 2^n$ locations
- All locations the same size
  - $2^n \times m$ bit memory



Memory Sizes

- Use power-of-2 multipliers
  - Kilo (K): $2^{10} = 1{,}024 \approx 10^3$
  - Mega (M): $2^{20} = 1{,}048{,}576 \approx 10^6$
  - Giga (G): $2^{30} = 1{,}073{,}741{,}824 \approx 10^9$
- Example
  - 32K × 32-bit memory
  - Capacity = 1,025K = 1Mbit
  - Requires 15 address bits
- Size is determined by application requirements

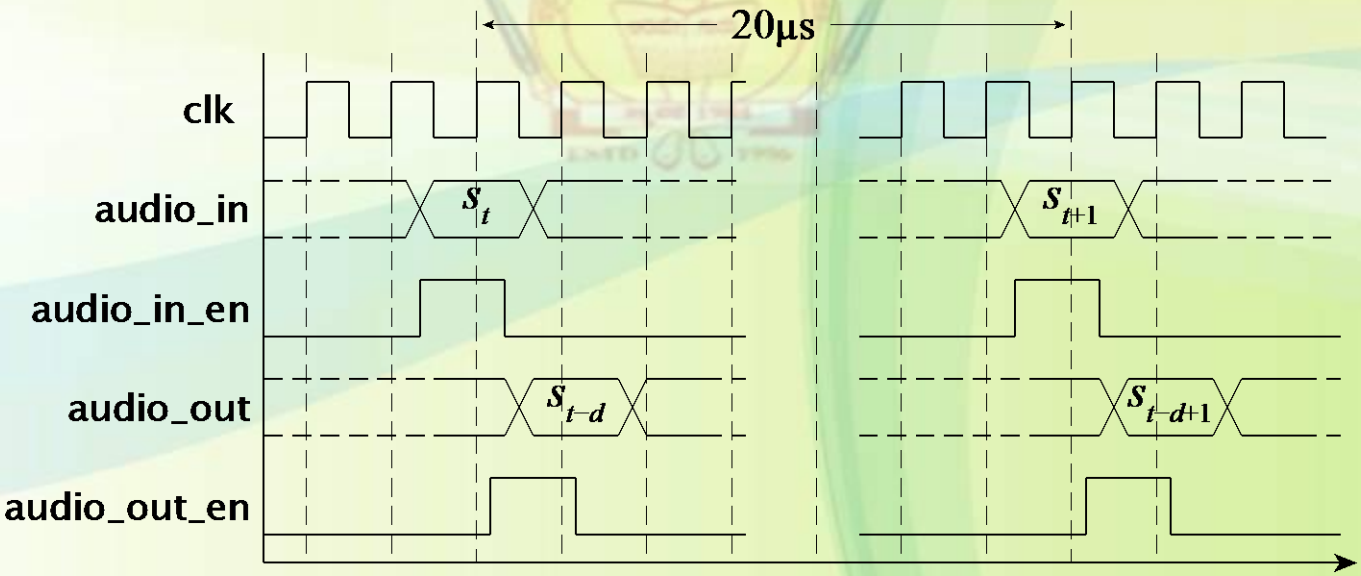
Basic Memory Operations



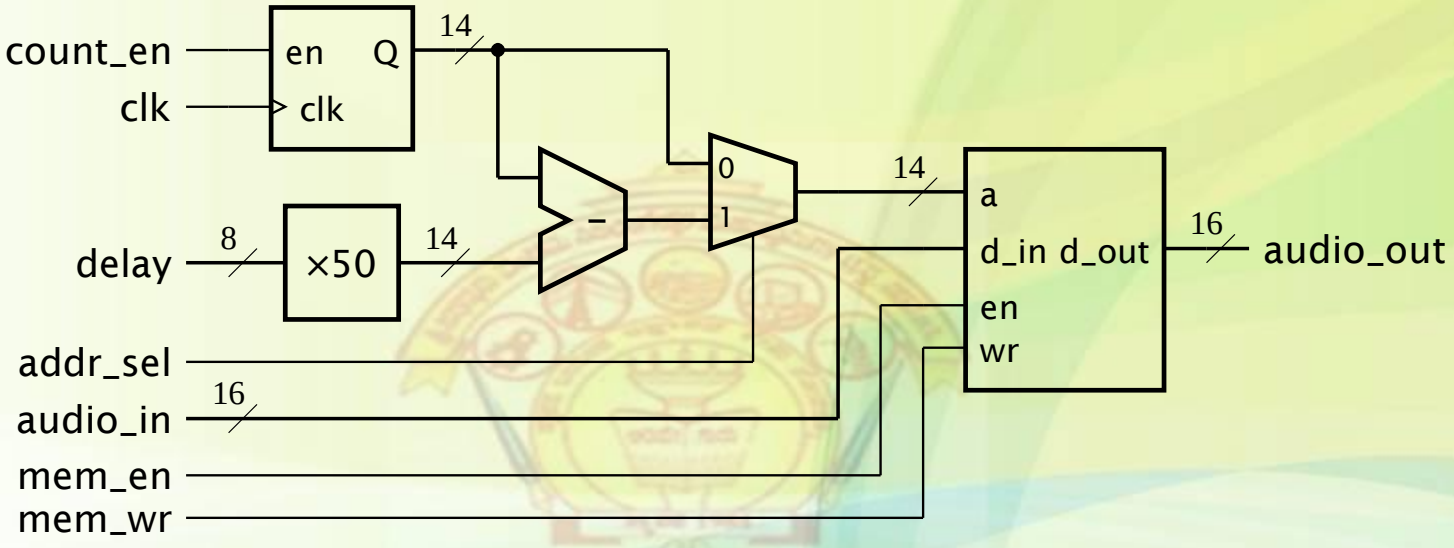
- a inputs: unsigned address
- d_in and d_out
  - Type depends on application
- Write operation
  - en = 1, wr = 1
  - d_in value stored in location given by address inputs
- Read operation
  - en = 1, wr = 0
  - d_out driven with value of location given by address inputs
- Idle: en = 0

Example: Audio Delay Unit

- System clock: 1MHz
- Audio samples: 8-bit signed, at 50kHz
  - New sample arrives when audio_in_en = 1
- Delay control: 8-bit unsigned ⇒ ms to delay
- Output: audio_out_en = 1 when output ready



Audio Delay Datapath



- Max delay = 255ms
  - Need to store 255 × 50 = 12,750 samples
  - Use a 16K × 8-bit memory (14 address bits)

Audio Delay Control Section

Step 1: (idle state)

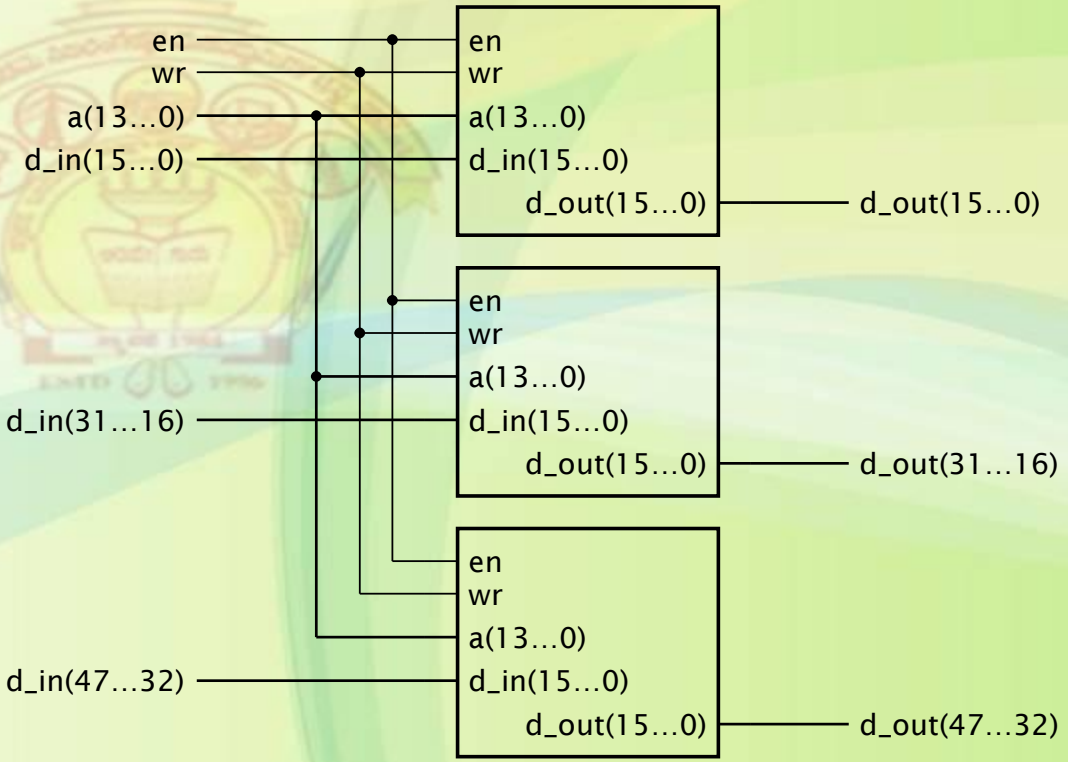
- audio_in_en = 0 ⇒ do nothing
- audio_in_en = 1 ⇒ write memory using counter value as address

- Step 2:
  - Read memory using subtracter output as address, increment counter

State	audio_ in_en	Next state	addr_sel	mem_en	mem_wr	count_en	audio_ out_en
Step 1	0	Step 1	0	0	0	0	0
Step 1	1	Step 2	0	1	1	0	0
Step 2	–	Step 1	1	1	0	1	1

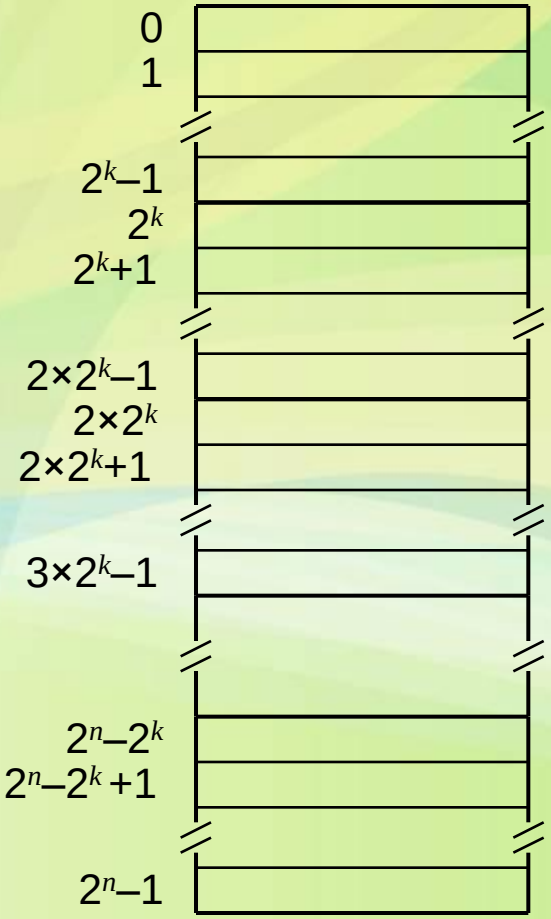
Wider Memories

- Memory components have a fixed width
  - E.g., ×1, ×4, ×8, ×16, ...
- Use memory components in parallel to make a wider memory
  - E.g, three 16K×16 components for a 16K×48 memory



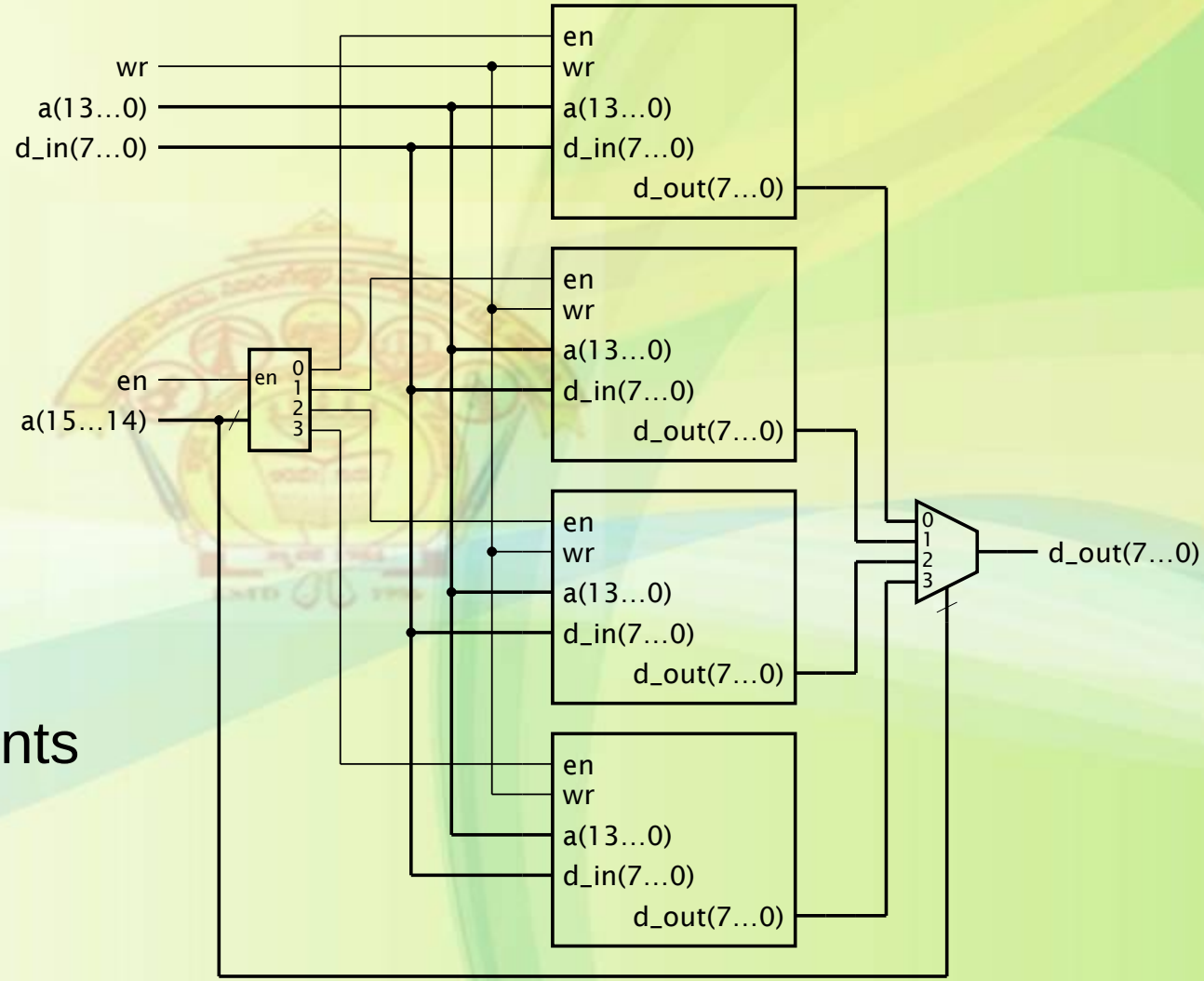
More Locations

- To provide 2^n locations with 2^k-location components
  - Use $2^n/2^k$ components
- Address A
  - at offset A mod 2^k
    - least-significant k bits of A
  - in component $\lfloor A/2^k \rfloor$
    - most-significant $n–k$ bits of A
    - decode to select component



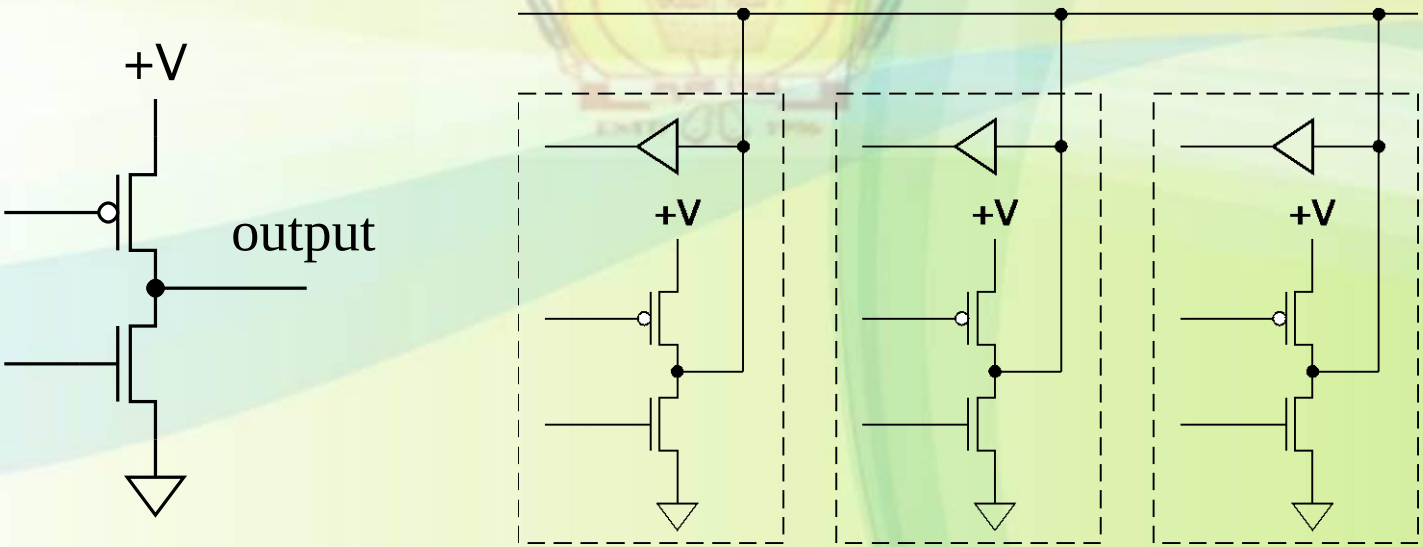
More Locations

- Example:
  64K×8 memory
  composed of
  16K×8 components



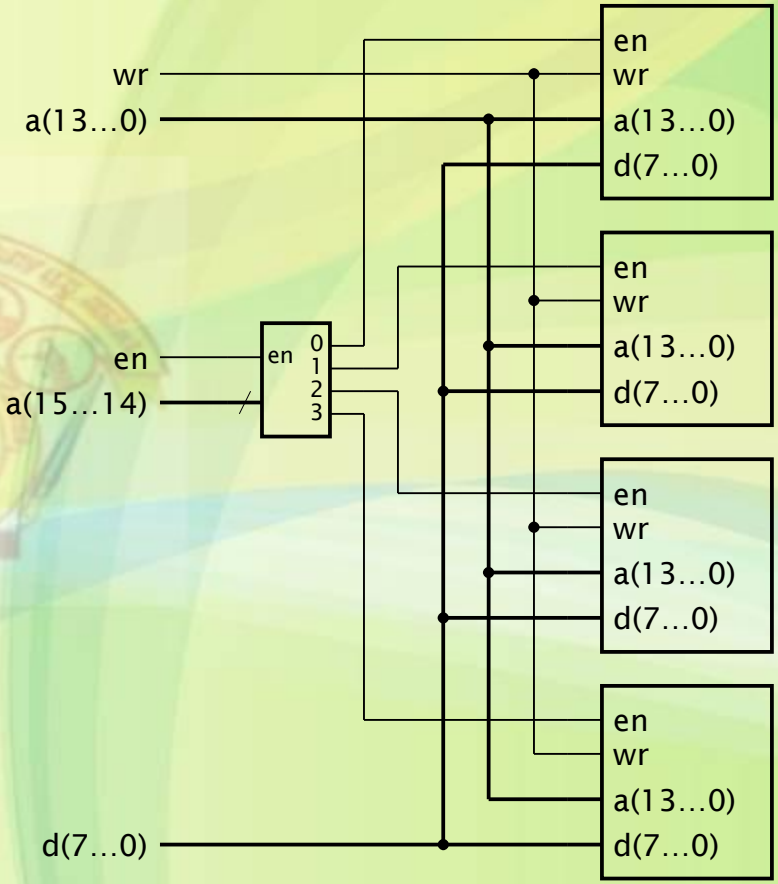
Tristate Drivers

- Allow multiple outputs to be connected together
  - Only one active at a time
  - Remaining outputs are high-impedance
    - Both output transistors turned off
- Allow bidirectional input/output ports



Memories with Tristate Ports

- During write
  - memory d drivers hi-Z
  - memory senses d
- During read
  - selected memory drives d
- Fewer pins and wires
  - Reduced cost of PCB
- Usually not available within ASICs or FPGAs

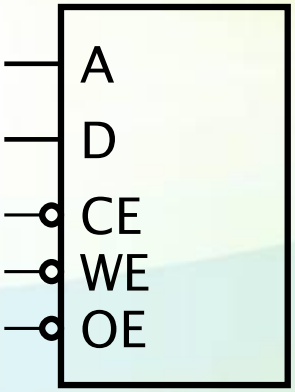


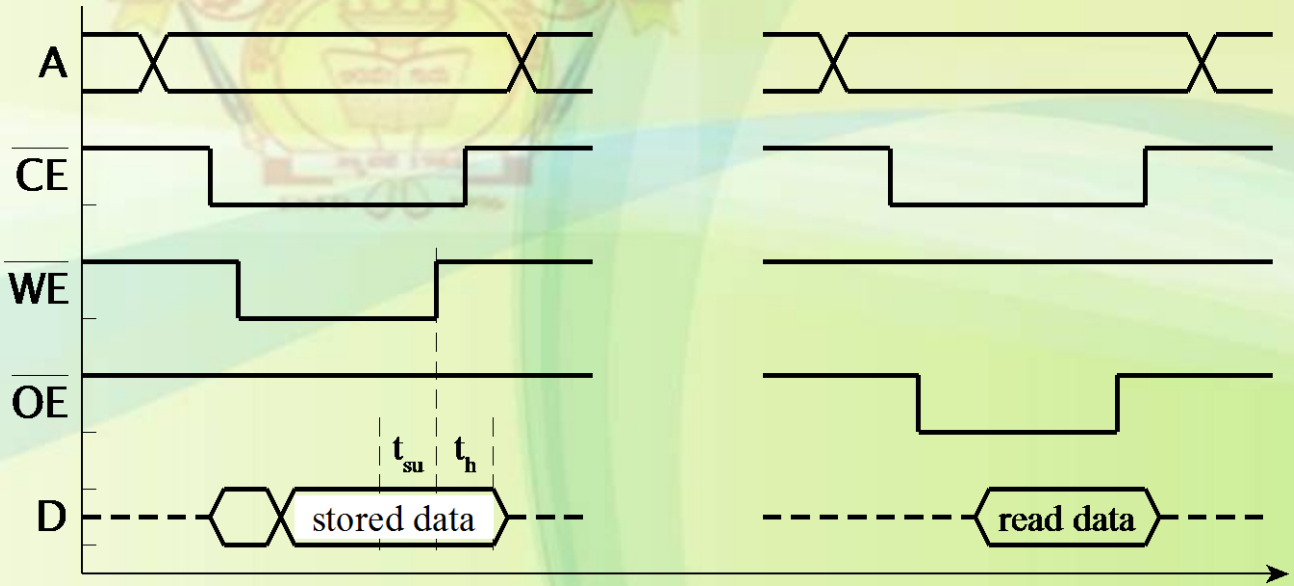
Memory Types

- Random-Access Memory (RAM)
  - Can read and write
  - Static RAM (SRAM)
    - Stores data so long as power is supplied
    - Asynchronous SRAM: not clocked
    - Synchronous SRAM (SSRAM): clocked
  - Dynamic RAM (DRAM)
    - Needs to be periodically refreshed
- Read-Only Memory (ROM)
  - Combinational
  - Programmable and Flash rewritable
- Volatile and non-volatile

Asynchronous SRAM

- Data stored in 1-bit latch cells
  - Address decoded to enable a given cell
- Usually use active-low control inputs
- Not available as components in ASICs or FPGAs





Asynch SRAM Timing

- Timing parameters published in data sheets
- Access time
  - From address/enable valid to data-out valid
- Cycle time
  - From start to end of access
- Data setup and hold
  - Before/after end of WE pulse
  - Makes asynch SRAMs hard to use in clocked synchronous designs

Example Data Sheet

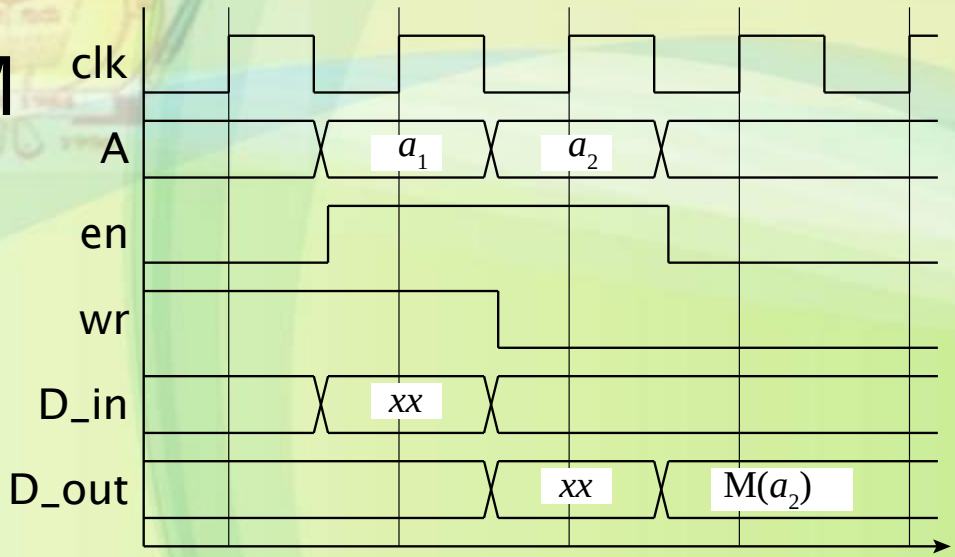
Switching Characteristics[4] Over the Operating Range

Parameter	Description	-12		-15		-17		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	
READ CYCLE								
t_{RC}	Read Cycle Time	12		15		17		ns
t_{AA}	Address to Data Valid		12		15		17	ns
t_{OHA}	Data Hold from Address Change	3		3		3		ns
t_{ACE}	$\overline{CE}$ LOW to Data Valid		12		15		17	ns
t_{DOE}	$\overline{OE}$ LOW to Data Valid		6		7		8	ns
WRITE CYCLE[7, 8]								
t_{WC}	Write Cycle Time	12		15		17		ns
t_{SCE}	$\overline{CE}$ LOW to Write End	10		12		12		ns
t_{AW}	Address Set-Up to Write End	10		12		12		ns
t_{HA}	Address Hold from Write End	0		0		0		ns
t_{SA}	Address Set-Up to Write Start	0		0		0		ns
t_{PWE}	$\overline{WE}$ Pulse Width	10		12		12		ns
t_{SD}	Data Set-Up to Write End	7		8		9		ns
t_{HD}	Data Hold from Write End	0		0		0		ns
t_{LZWE}	$\overline{WE}$ HIGH to Low Z[6]	3		3		3		ns
t_{HZWE}	$\overline{WE}$ LOW to High Z[5, 6]		6		7		8	ns
t_{BW}	Byte Enable to End of Write	10		12		12		ns

Synchronous SRAM (SSRAM)

- Clocked storage registers for inputs
  - address, data and control inputs
  - stored on a clock edge
  - held for read/write cycle

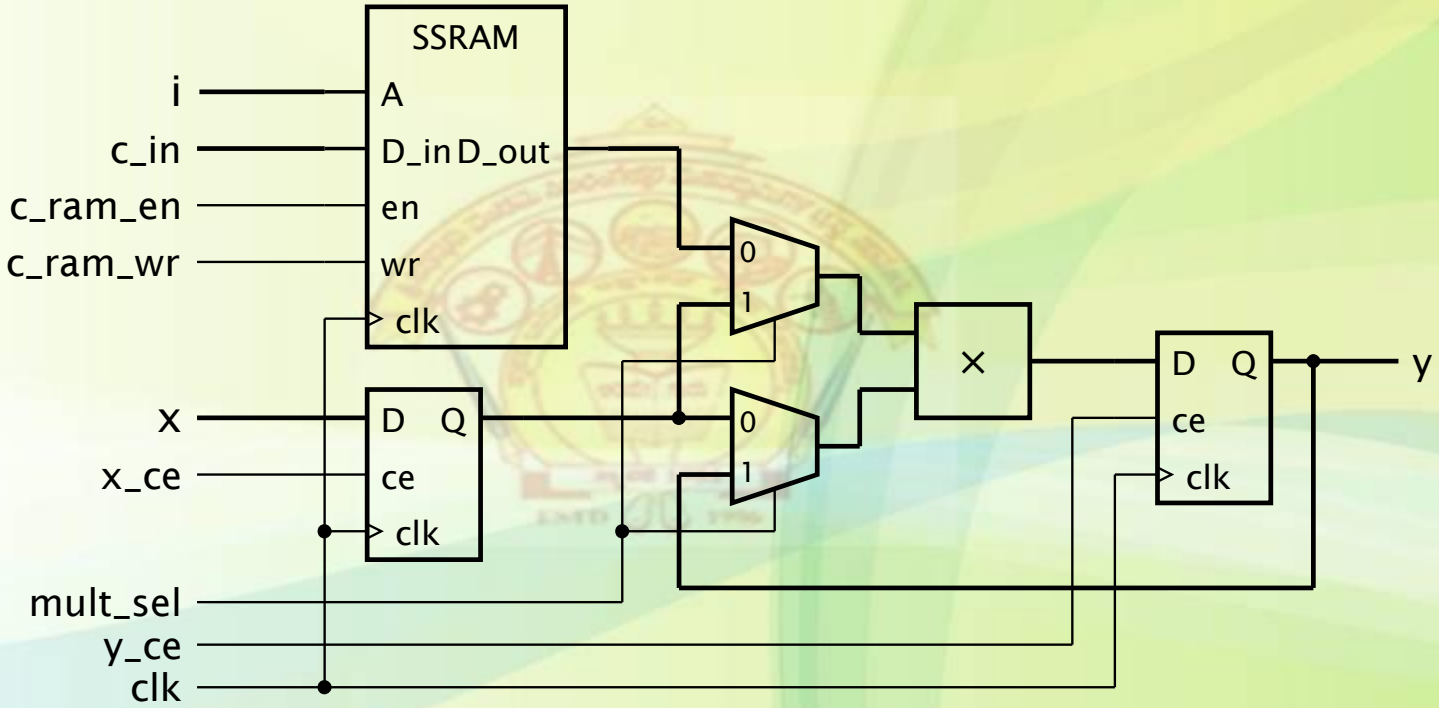
- Flow-through SSRAM
  - no register on data output



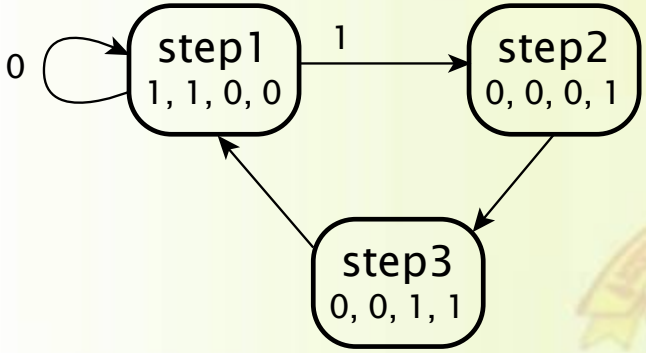
Example: Coefficient Multiplier

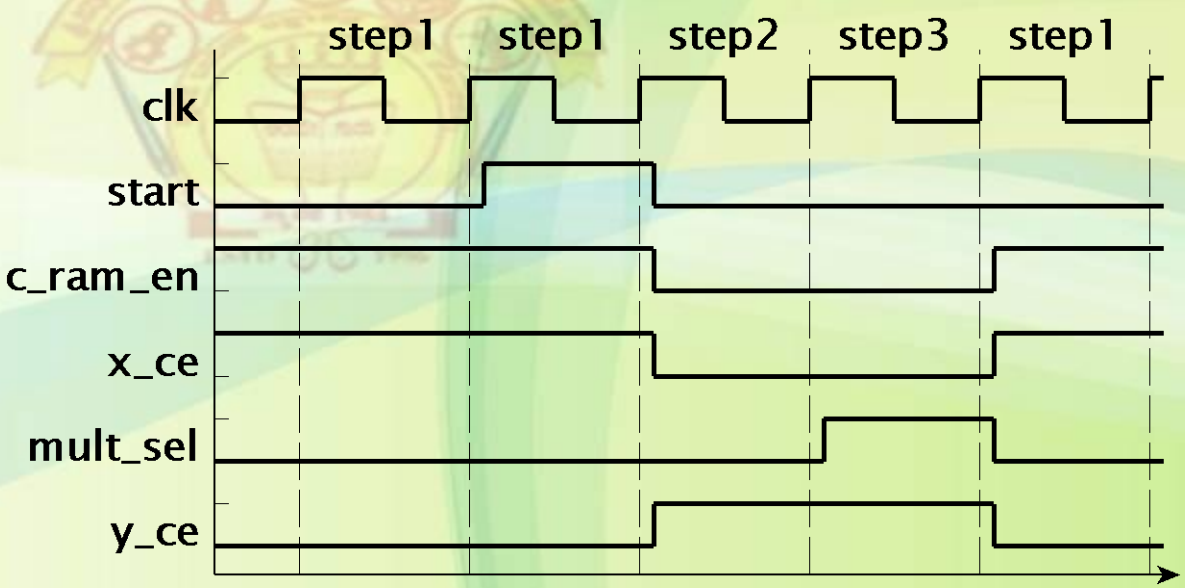
- Compute function $y = c_i \times x^2$
  - Coefficient stored in flow-through SSRAM
    - 12-bit unsigned integer index for i
  - x, y, c_i 20-bit signed fixed-point
    - 8 pre- and 12 post-binary point bits
  - Use a single multiplier
    - Multiply $c_i \times x \times x$

Multiplier Datapath



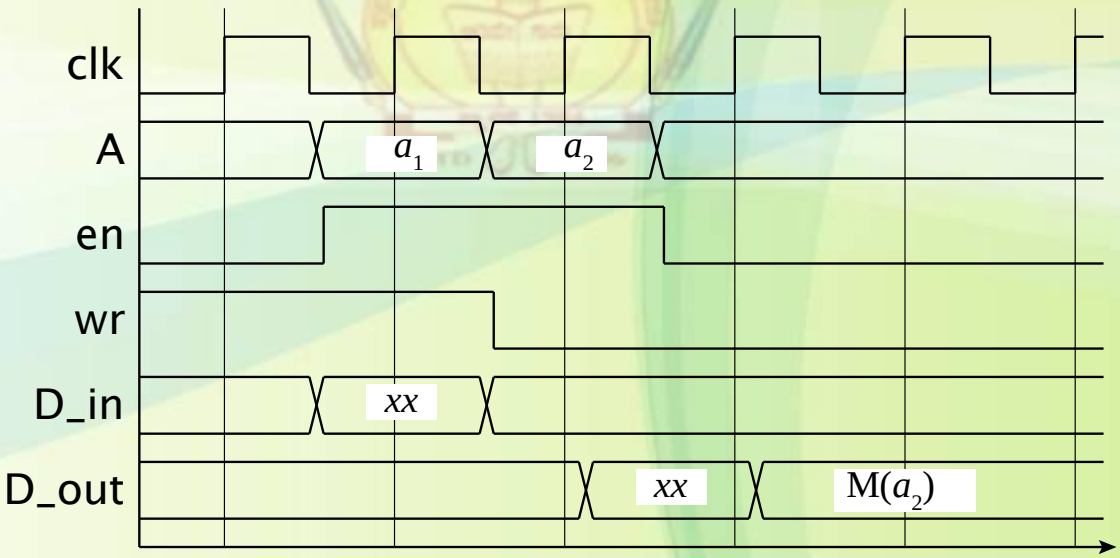
Multiplier Timing and Control





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
```

Pipelined SSRAM in Verilog

```
reg         pipelined_en;
reg [15:0] pipelined_d_out;
...

always @(posedge clk) begin
  if (pipelined_en) d_out <= pipelined_d_out;
  pipelined_en <= en;
  if (en)
    if (wr) begin
      data_RAM([a] <= d_in;  pipelined_d_out <= d_in;
    end
    else
      pipelined_d_out <= data_RAM[a];
end
```

output register

SSRAM

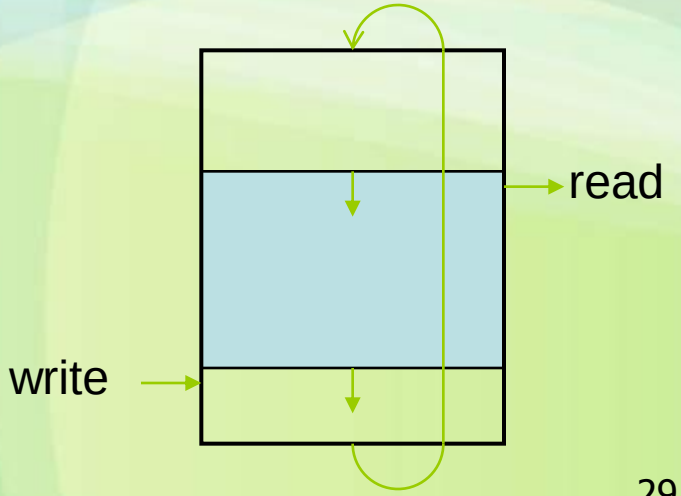
Multiport Memories

- 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

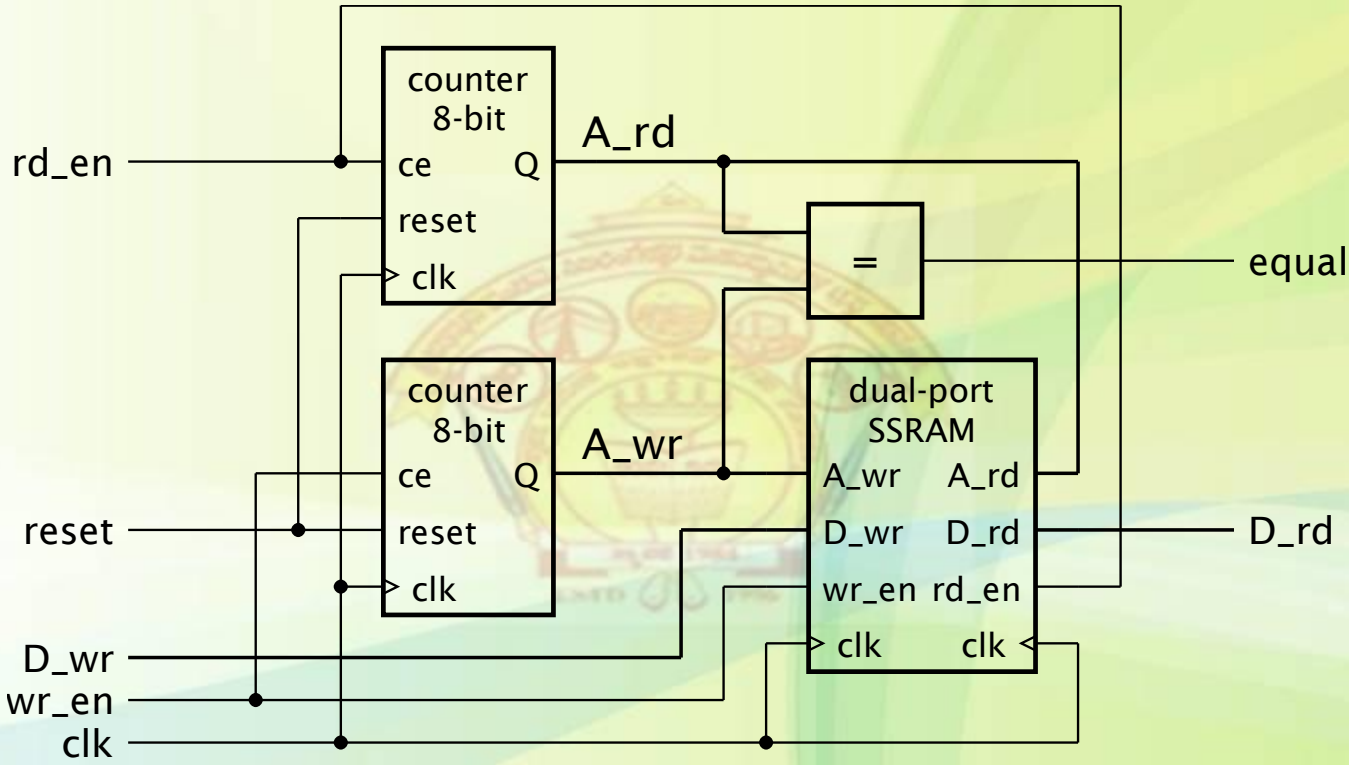
FIFO Memories

- 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



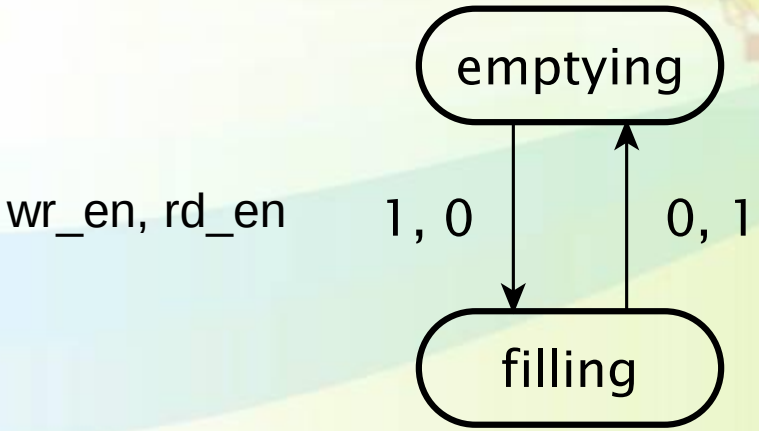
Example: FIFO Datapath



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

Example: FIFO Control

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



full = filling and equal

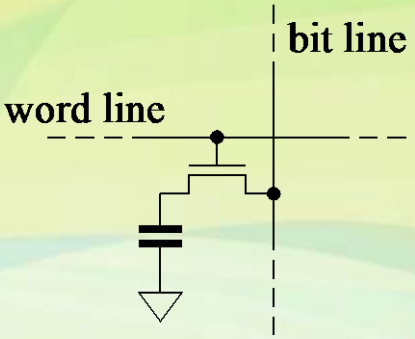
empty = emptying and equal

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

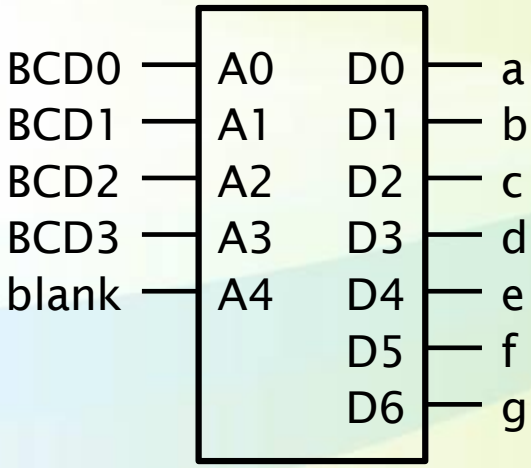
- Charge on capacitor decays over time
  - Need to sense and rewrite periodically
    - Typically every cell every 64ms
  - *Refresh* each location
- DRAMs organized into banks of rows
  - Refresh whole row at a time
- Can’t access while refreshing
  - Interleave refresh among accesses
  - Or burst refresh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

Memory Errors

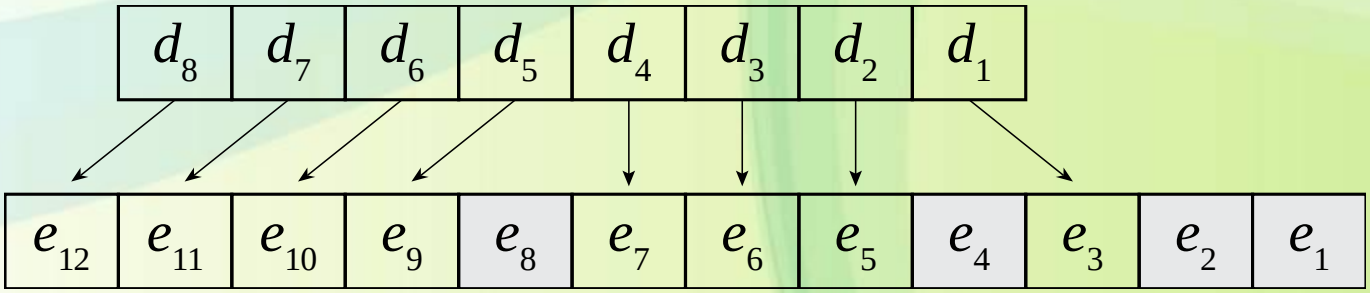
- Bits in memory can be flipped
- Hard error
  - The chip is broken
  - E.g., manufacturing defect, wear (in Flash)
- Soft error
  - Stored data corrupted, but cell still works
  - E.g., from atmospheric neutrons
- Soft-error rate
  - frequency of occurrence

Error Detection using Parity

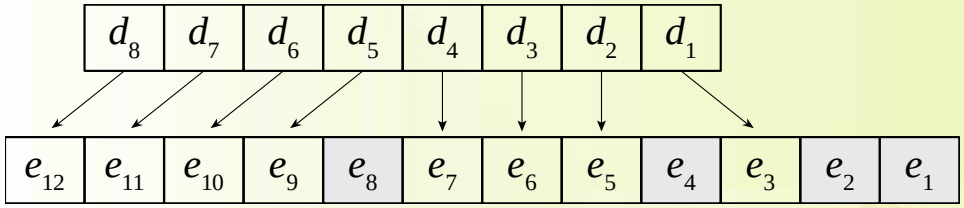
- Add a parity bit to each location
- On write access
  - compute data parity and store with data
- On read access
  - check parity, take exception on error
- If we could tell which bit flipped
  - correct by flipping it back, then write back to memory location
  - Can’t do this with parity

Error-Correcting Codes (ECC)

- Allow identification of the flipped bit
- Hamming Codes
  - E.g., for single-bit-error correction of *N*-bit word, need $\log_2 N + 1$ extra bits
- Example: 8-bit word, $d_1 \ldots d_8$
  - 12-bit ECC code, $e_1 \ldots e_{12}$
  - e_1, e_2, e_4, e_8 are check bits, the rest data



Hamming Code Example



e_1	0	0	0	1
e_2	0	0	1	0
e_4	0	1	0	0
e_8	1	0	0	0
e_3	0	0	1	1
e_5	0	1	0	1
e_6	0	1	1	0
e_7	0	1	1	1
e_9	1	0	0	1
e_{10}	1	0	1	0
e_{11}	1	0	1	1
e_{12}	1	1	0	0

$$e_1 = e_3 \oplus e_5 \oplus e_7 \oplus e_9 \oplus e_{11}$$

$$e_2 = e_3 \oplus e_6 \oplus e_7 \oplus e_{10} \oplus e_{11}$$

$$e_4 = e_5 \oplus e_6 \oplus e_7 \oplus e_{12}$$

$$e_8 = e_9 \oplus e_{10} \oplus e_{11} \oplus e_{12}$$

- Every data bit covered by two or more check bits
- On write: Compute check bits and store with data

Hamming Code Example

e_1	0	0	0	1
e_2	0	0	1	0
e_4	0	1	0	0
e_8	1	0	0	0
e_3	0	0	1	1
e_5	0	1	0	1
e_6	0	1	1	0
e_7	0	1	1	1
e_9	1	0	0	1
e_{10}	1	0	1	0
e_{11}	1	0	1	1
e_{12}	1	1	0	0

- On read: Recompute check bits and XOR with read check bits
  - result called the *syndrome*
- 0000 => no error
- If data bit flipped
  - covering bits of syndrome are 1
  - = binary code of flipped ECC bit
- If stored check bit flipped
  - that bit of syndrome is 1
- On error, unflip bit and rewrite memory location

Multiple-Error Detection

- What if two bits flip
  - syndrome identifies wrong bit, or is invalid
- One extra check bit allows
  - single-error correction, double-error detection

N	Single-bit correction		Double-bit detection	
	Check bits	Overhead	Check bits	Overhead
8	4	50%	5	63%
16	5	31%	6	38%
32	6	19%	7	22%
64	7	11%	8	13%
128	8	6.3%	9	7.0%
256	9	3.5%	10	3.9%

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
- Error Detection and Correction
  - Hamming Codes

Queries ...?