E85 Digital Design & Computer Engineering

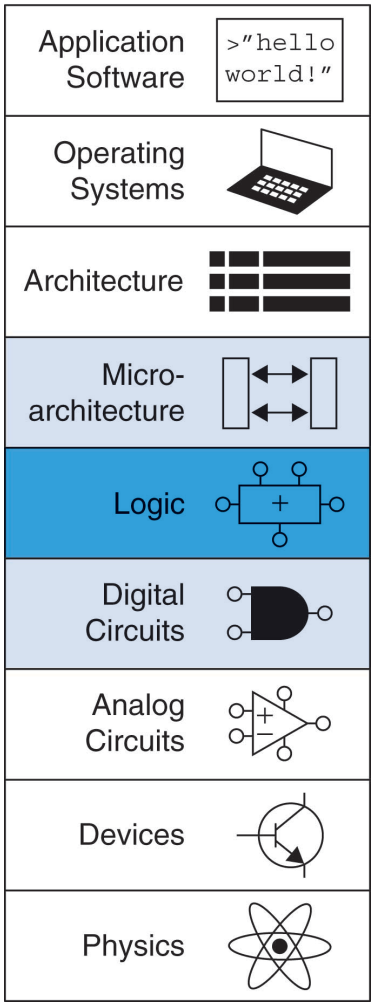
Lecture 10:

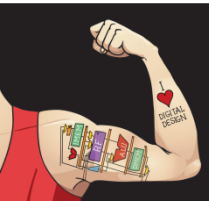
Sequential Blocks
Arrays

HARVEY MUDD COLLEGE

Lecture 8

- Counters
- Shift Registers
- Memory Arrays
  - RAM
  - ROM
- Logic Arrays
  - PLAs
  - FPGAs

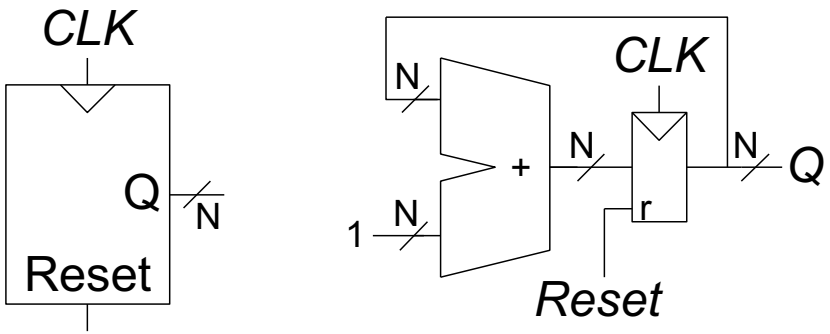




Counters

- Increments on each clock edge
- Used to cycle through numbers. For example,
  - 000, 001, 010, 011, 100, 101, 110, 111, 000, 001…
- Example uses:
  - Digital clock displays
  - Program counter: keeps track of current instruction executing

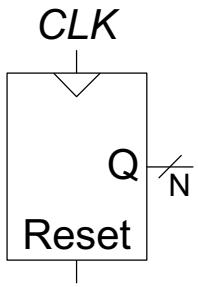
Symbol **Implementation**



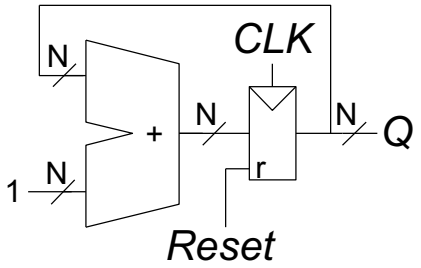
Counter Verilog (FSM style)

```
module counter (input  logic          clk, reset,
                output logic [N-1:0] q);
  logic [N-1:0] nextq;

  // register
  always_ff @(posedge clk, posedge reset)
    if (reset) q <= 0;
    else       q <= nextq;

  // next state
  assign nextq = q + 1;
endmodule
```

Symbol



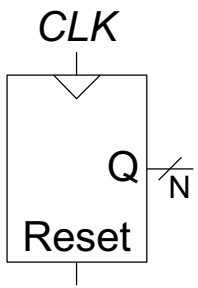
Implementation



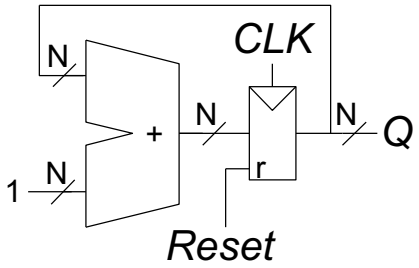
Counter Verilog (better idiom)

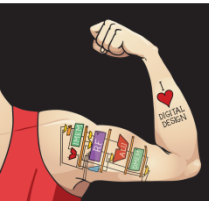
```
module counter (input  logic          clk, reset,
                output logic [N-1:0] q);
always_ff @(posedge clk, posedge reset)
    if (reset) q <= 0;
    else       q <= q+1;
endmodule
```

Symbol



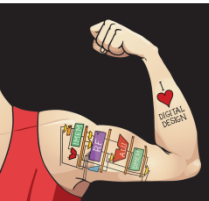
Implementation





Divide-by-2^N Counter

- Most significant bit of an N-bit counter toggles every 2^N cycles.
- Useful for slowing a clock. Ex: blink an LED
- Example: 50 MHz clock, 24-bit counter
  - 2.98 Hz



Digitally Controlled Oscillator

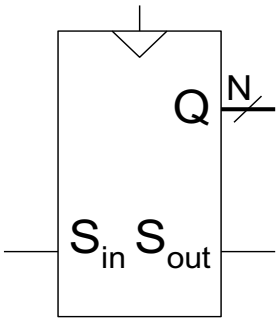
- N-bit counter
- Add p on each cycle, instead of 1
- Most significant bit toggles at $f_{out} = f_{clk} * p / 2^N$

- Example: f_{clk} = 50 MHz clock
  - How to generate a f_{out} = 200 Hz signal?
  - $p/2^N = 200 / 50$ MHz
- Try N = 24, p = 67 ➔ f_{out} = 199.676 Hz
- Or N = 32, p = 17179 ➔ f_{out} = 199.990 Hz

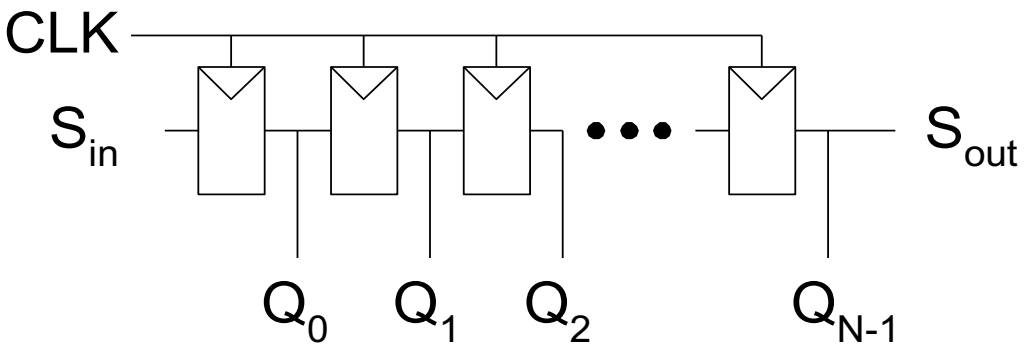
Shift Registers

- Shift a new bit in on each clock edge
- Shift a bit out on each clock edge
- *Serial-to-parallel converter*: converts serial input (S_{in}) to parallel output ($Q_{0:N-1}$)

Symbol:

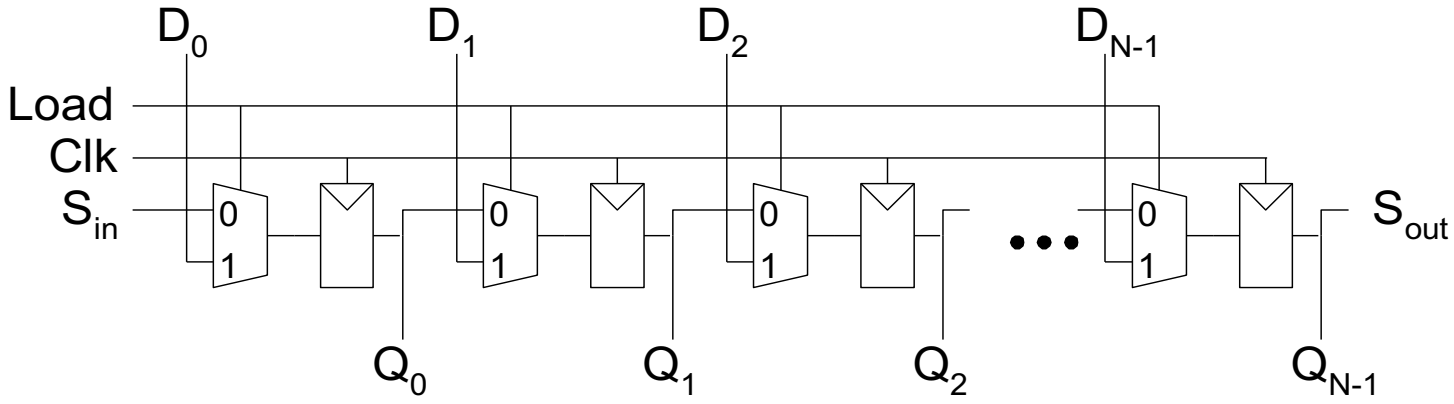


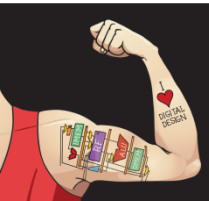
Implementation:



Shift Register with Parallel Load

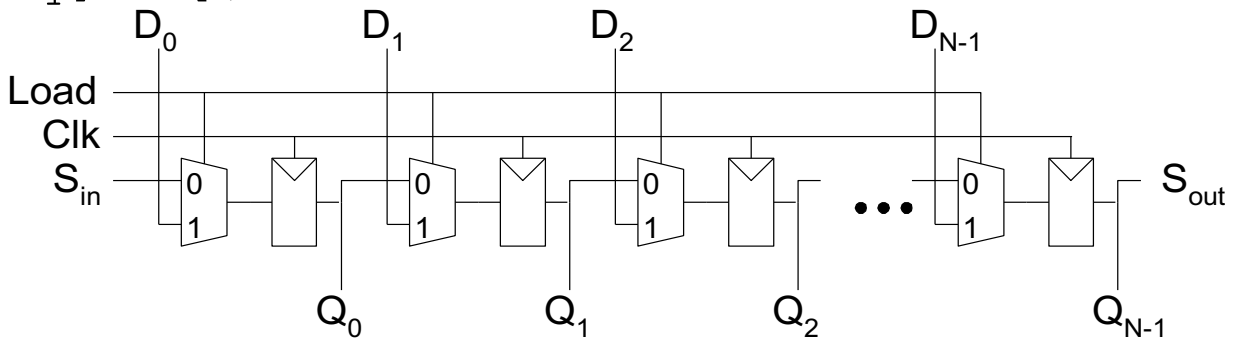
- When *Load* = 1, acts as a normal *N*-bit register
- When *Load* = 0, acts as a shift register
- Now can act as a *serial-to-parallel converter* (S_{in} to $Q_{0:N-1}$) or a *parallel-to-serial converter* ($D_{0:N-1}$ to S_{out})

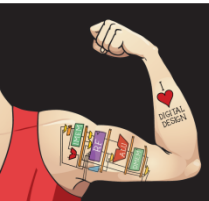




Shift Register Verilog Idiom

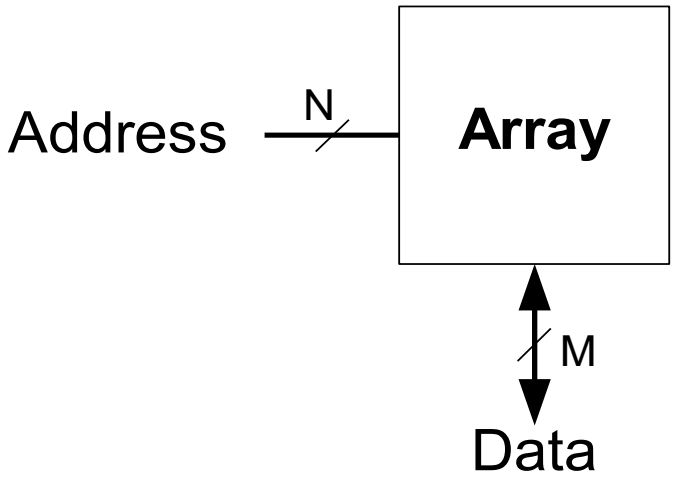
```
module shiftreg(input  logic           clk,
                input  logic           load, sin,
                input  logic [N-1:0]   d,
                output logic [N-1:0]   q,
                output logic           sout);
always_ff @(posedge clk)
    if (load)  q <= d;
    else       q <= {q[N-2:0], sin};
  assign sout = q[N-1];
endmodule
```





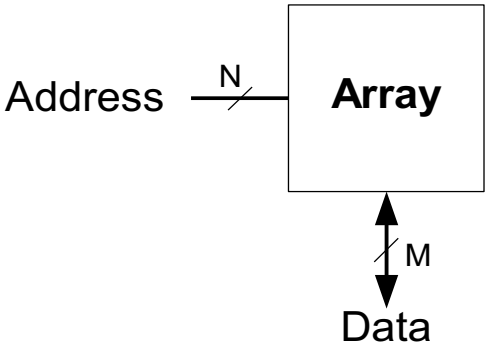
Memory Arrays

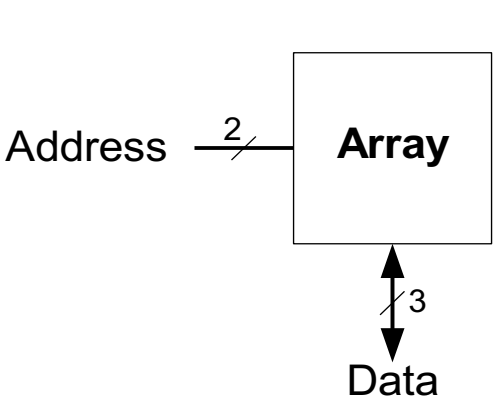
- Efficiently store large amounts of data
- 3 common types:
  - Dynamic random access memory (DRAM)
  - Static random access memory (SRAM)
  - Read only memory (ROM)
- *M*-bit data value read/ written at each unique *N*-bit address

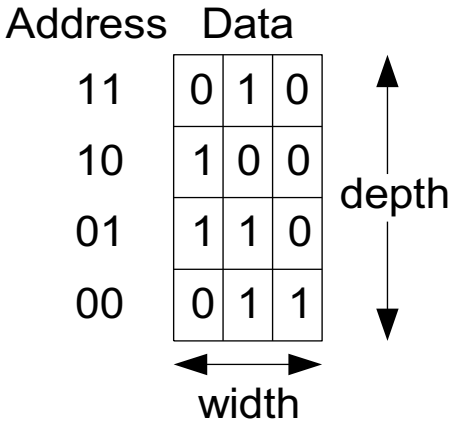


Memory Arrays

- 2-dimensional array of bit cells
- Each bit cell stores one bit
- *N* address bits and *M* data bits:
  - 2^N rows and *M* columns
  - **Depth:** number of rows (number of words)
  - **Width:** number of columns (size of word)
  - **Array size:** depth × width = $2^N \times M$

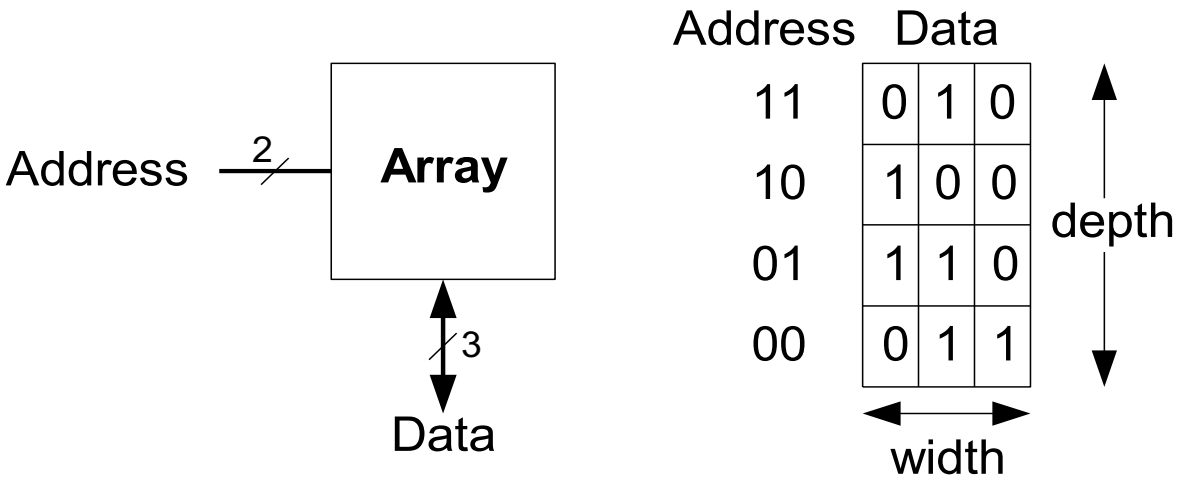




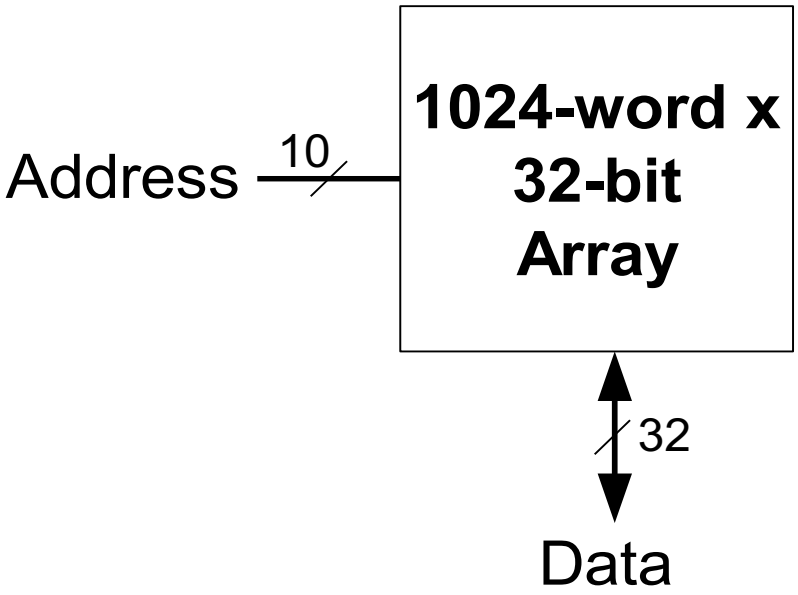


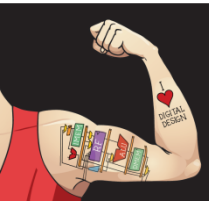
Memory Array Example

- $2^2 \times 3$-bit array
- Number of words: 4
- Word size: 3-bits
- For example, the 3-bit word stored at address 10 is 100

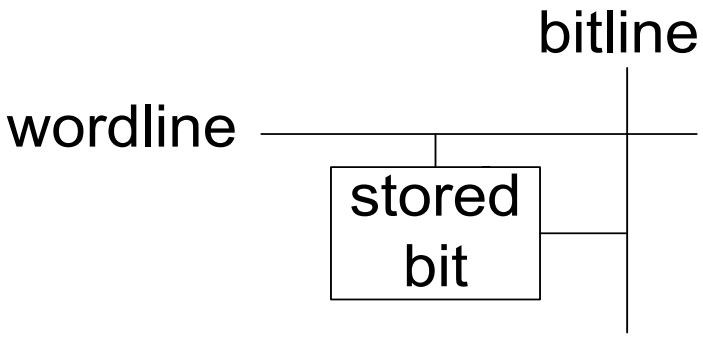


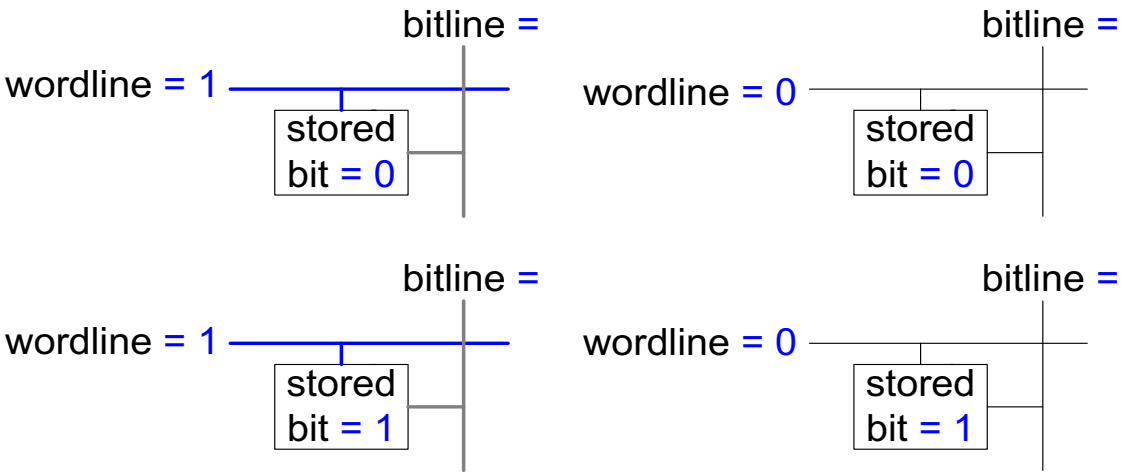
Memory Arrays

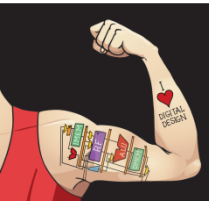




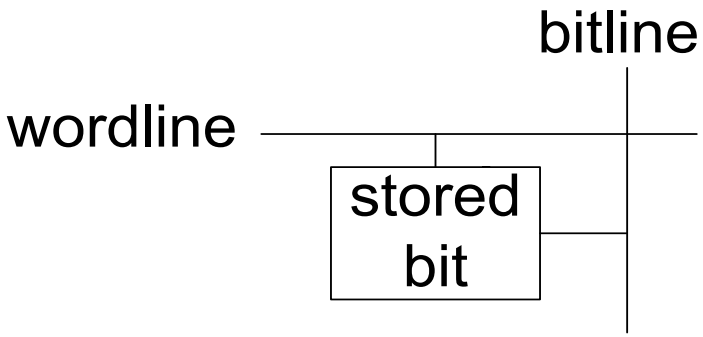
Memory Array Bit Cells

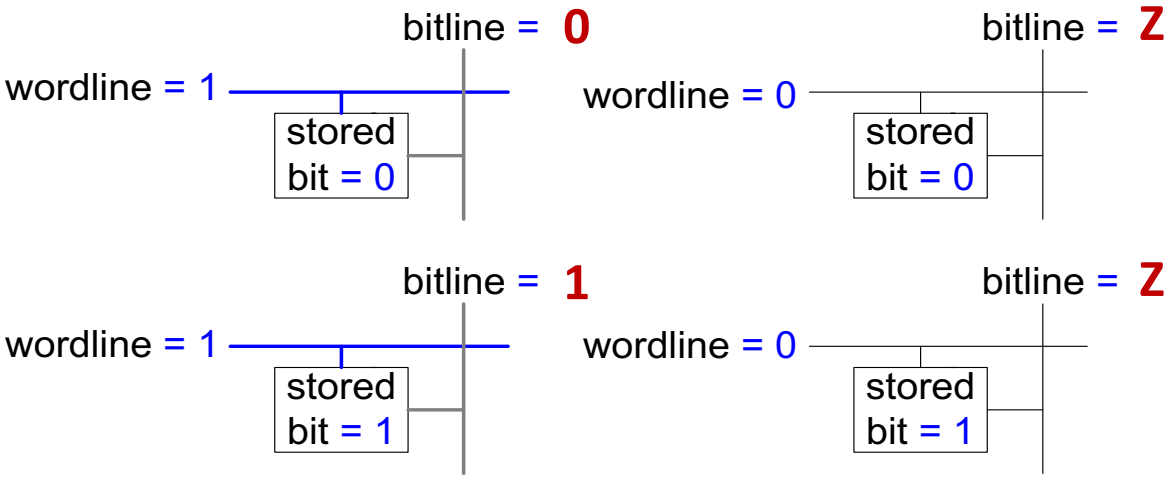


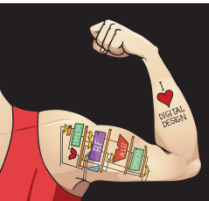




Memory Array Bit Cells

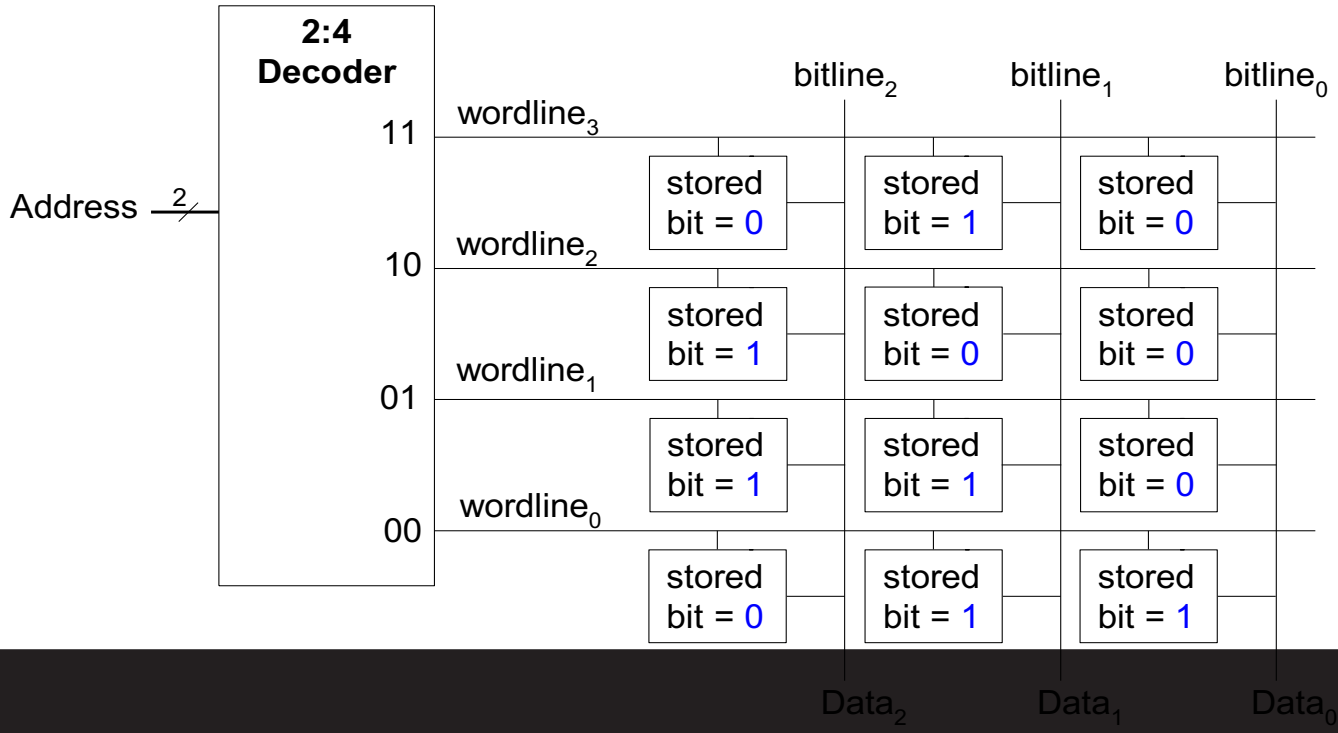






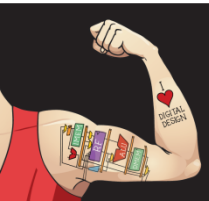
Memory Array

- **Wordline:**
  - like an enable
  - single row in memory array read/written
  - corresponds to unique address
  - only one wordline HIGH at once



Types of Memory

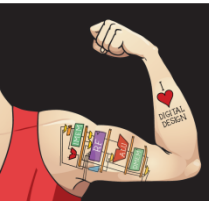
- Random access memory (RAM): **volatile**
- Read only memory (ROM): **nonvolatile**



RAM: Random Access Memory

- **Volatile:** loses its data when power off
- Read and written quickly
- Main memory in your computer is RAM (DRAM)

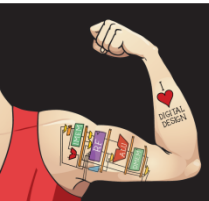
Historically called *random* access memory because any data word accessed as easily as any other (in contrast to sequential access memories such as a tape recorder)



ROM: Read Only Memory

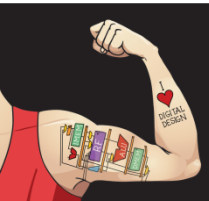
- **Nonvolatile:** retains data when power off
- Read quickly, but writing is impossible or slow
- Flash memory in cameras, thumb drives, and digital cameras are all ROMs

Historically called *read only* memory because ROMs were written at manufacturing time or by burning fuses. Once ROM was configured, it could not be written again. This is no longer the case for Flash memory and other types of ROMs.



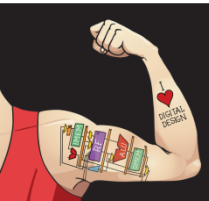
Types of RAM

- **DRAM** (Dynamic random access memory)
- **SRAM** (Static random access memory)
- Differ in how they store data:
  - DRAM uses a capacitor
  - SRAM uses cross-coupled inverters



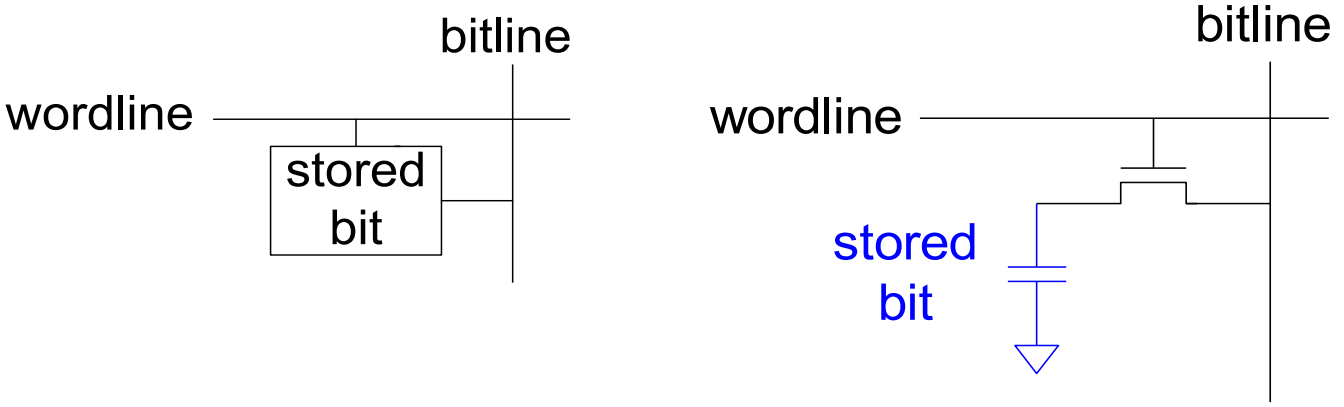
Robert Dennard, 1932 -

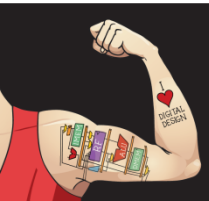
- Invented DRAM in 1966 at IBM
- Others were skeptical that the idea would work
- By the mid-1970's DRAM in virtually all computers



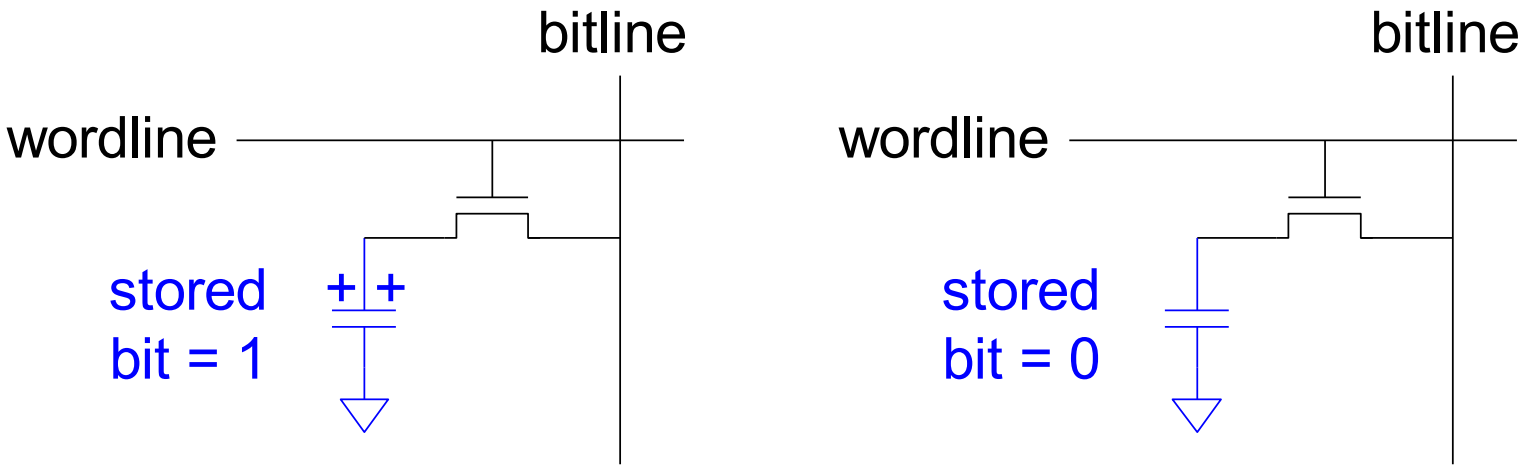
DRAM

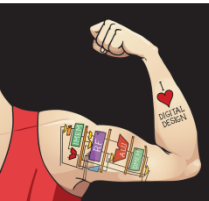
- Data bits stored on capacitor
- *Dynamic* because the value needs to be refreshed (rewritten) periodically and after read:
  - Charge leakage from the capacitor degrades the value
  - Reading destroys the stored value



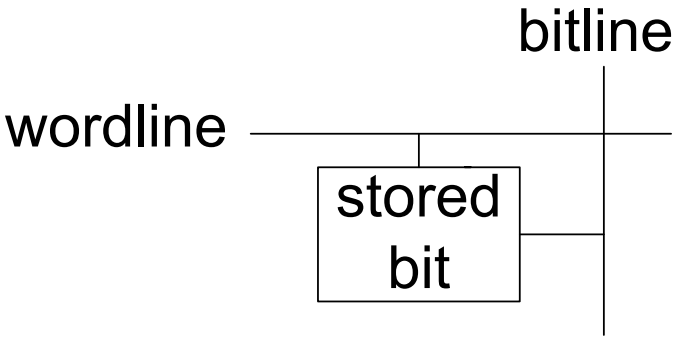


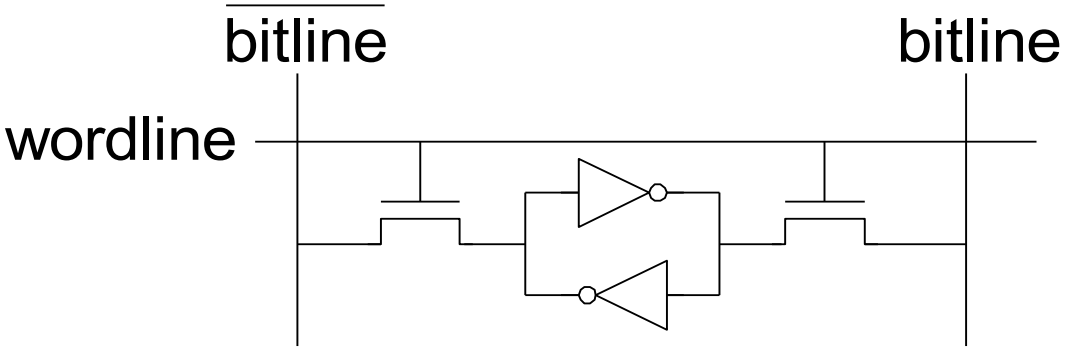
DRAM


bitline
wordline
stored
bit = 1
+ +
bitline
wordline
stored
bit = 0

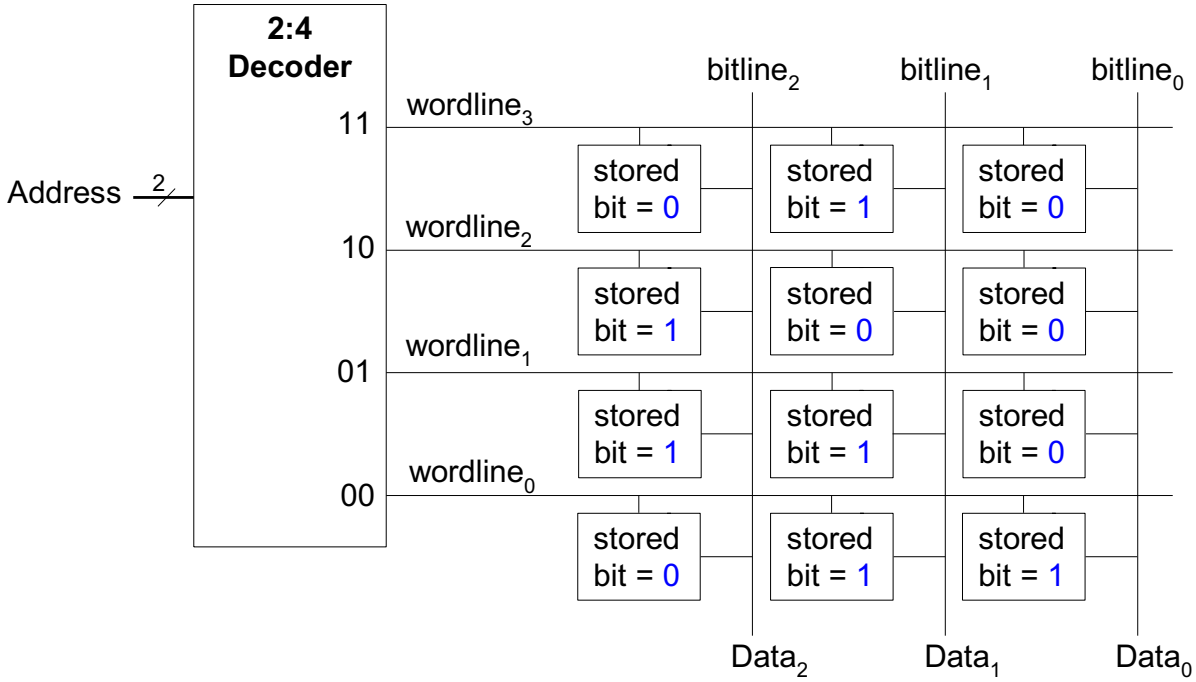


SRAM

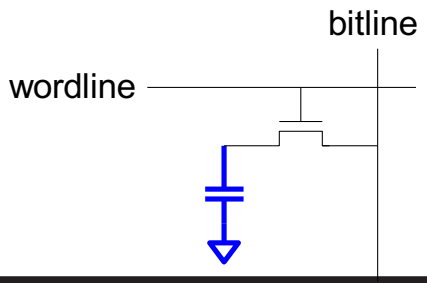




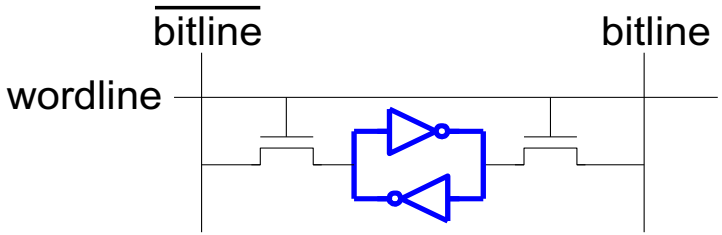
Memory Arrays Review

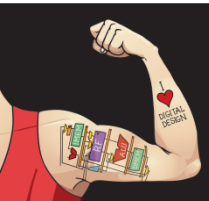


DRAM bit cell:

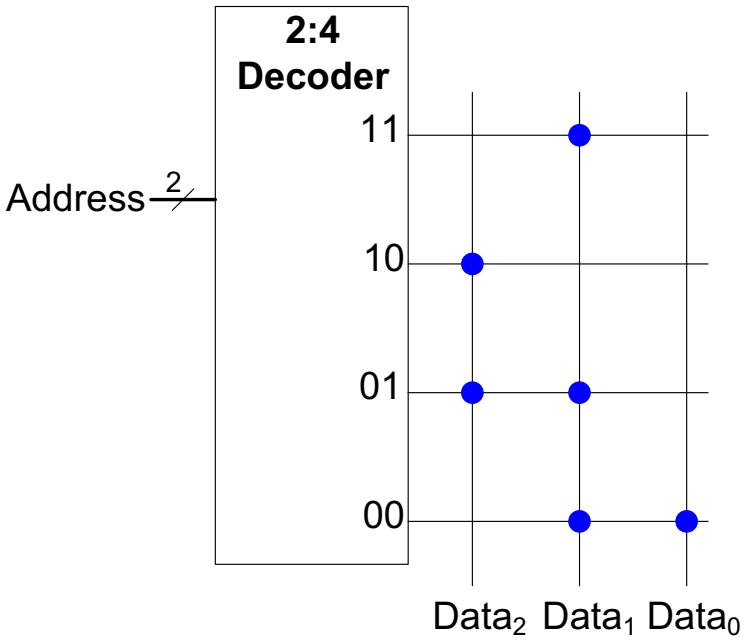


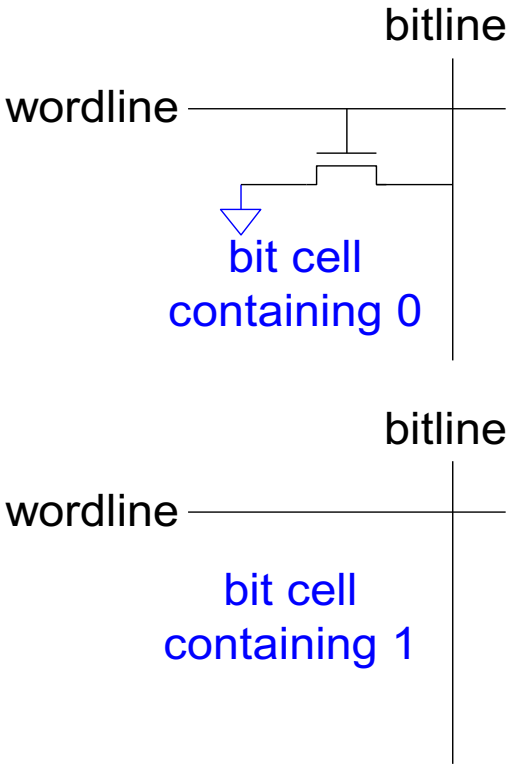
SRAM bit cell:





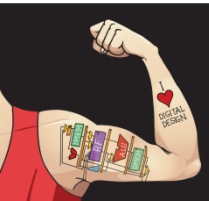
ROM: Dot Notation





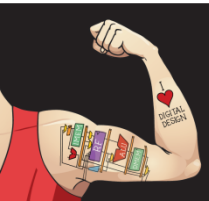
Types of ROMs

Type	Name	Description
ROM	Read Only Memory	Chip is hardwired with presence or absence of transistors. Changing requires building a new chip.
PROM	Programmable ROM	Fuses in series with each transistor are blown to program bits. Can't be changed after programming.
EPROM	Electrically Programmable ROM	Charge is stored on a floating gate to activate or deactivate transistor. Erasing requires exposure to UV light.
EEPROM	Electrically Erasable Programmable ROM	Like EPROM, but erasing can be done electrically.
Flash	Flash Memory	Like EEPROM, but erasing is done on large blocks to amortize cost of erase circuit. Low cost per bit, dominates nonvolatile storage today.

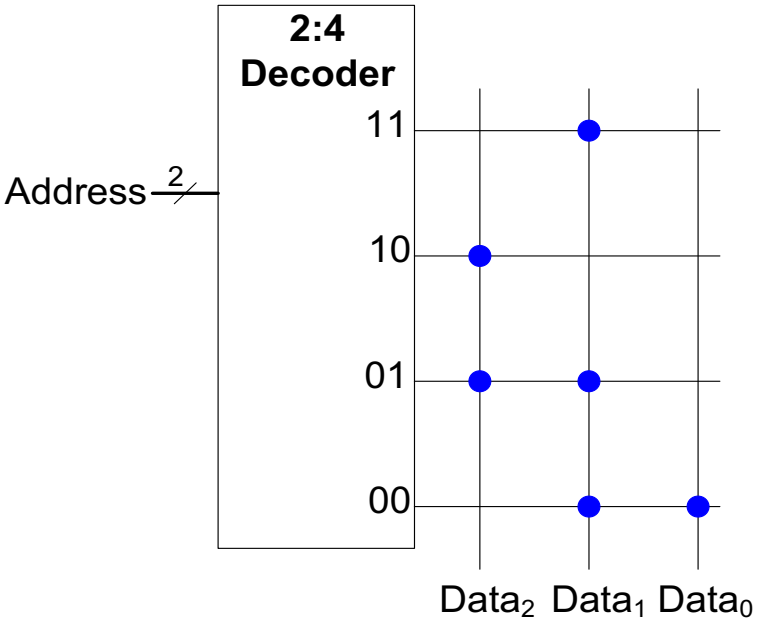


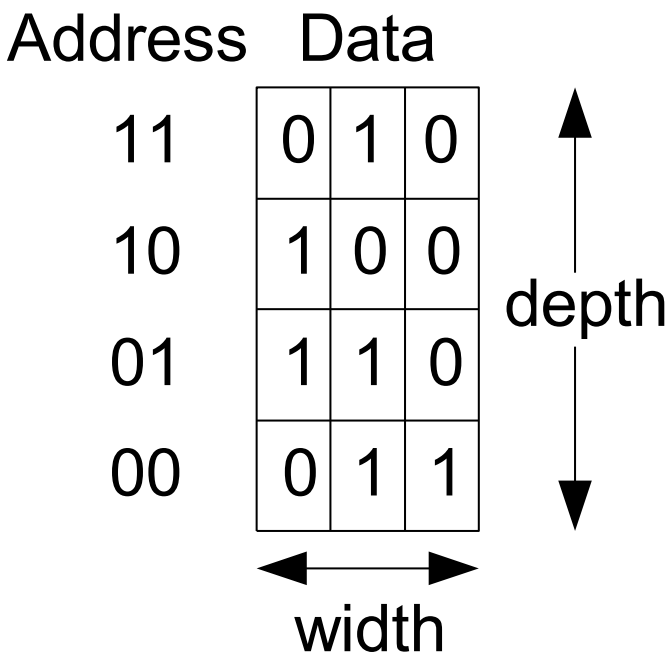
Fujio Masuoka, 1944 -

- Developed memories and high speed circuits at Toshiba, 1971-1994
- Invented Flash memory as an unauthorized project pursued during nights and weekends in the late 1970's
- The process of erasing the memory reminded him of the flash of a camera
- Toshiba slow to commercialize the idea; Intel was first to market in 1988
- Flash has grown into a $25 billion per year market

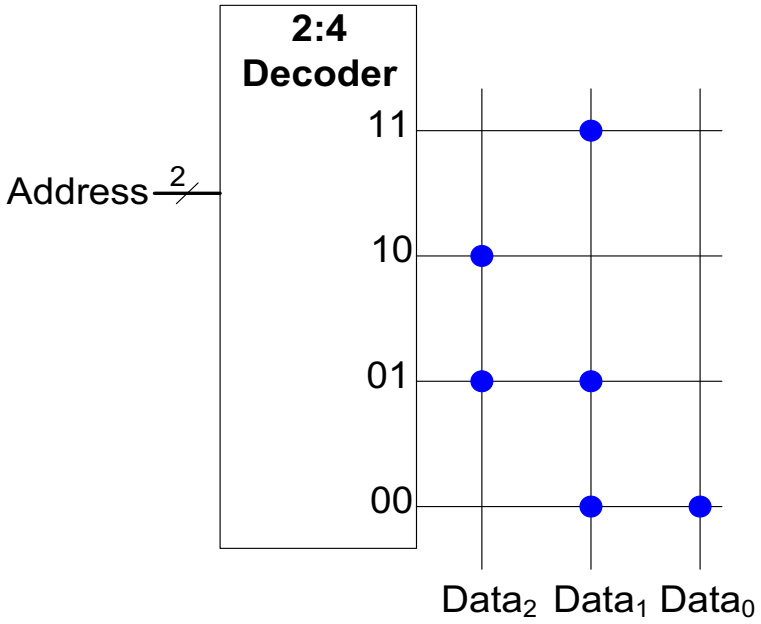


ROM Storage





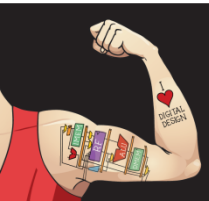
ROM Logic



$$Data_2 = A_1 \perp A_0$$

$$Data_1 = \overline{A_1} + A_0$$

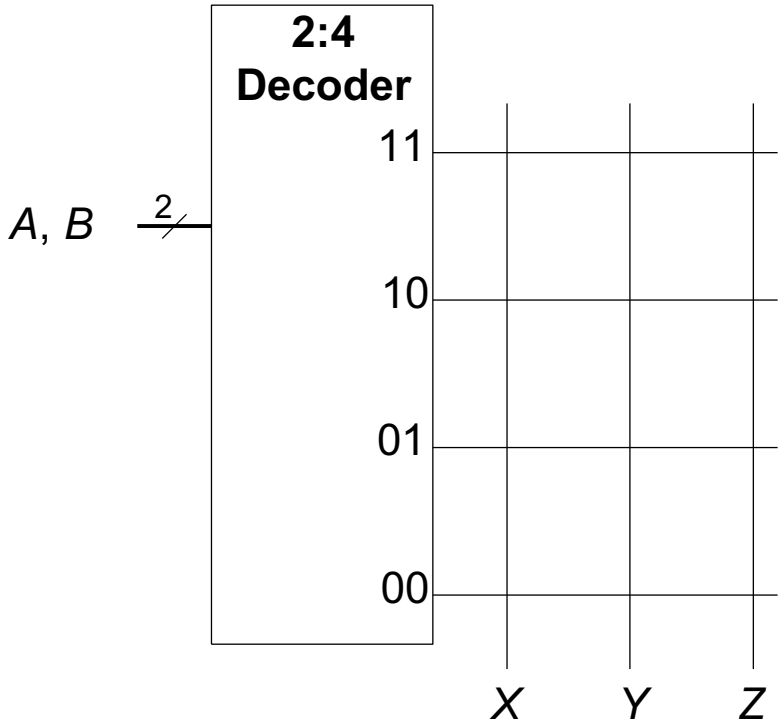
$$Data_0 = \overline{A_1}\,\overline{A_0}$$

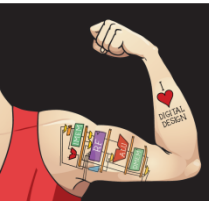


Example: Logic with ROMs

Implement the following logic functions using a $2^2 \times 3$-bit ROM:

- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$

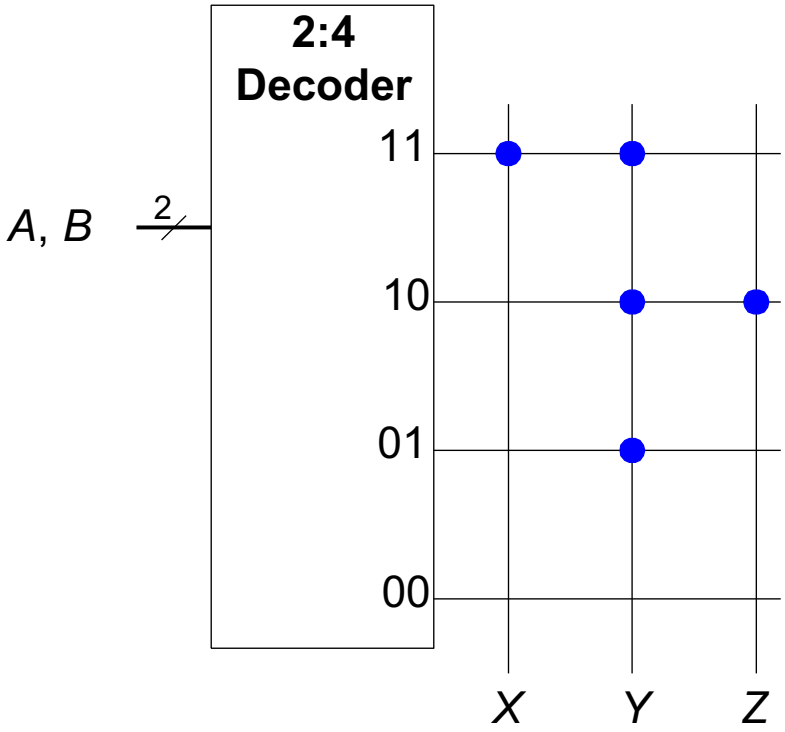




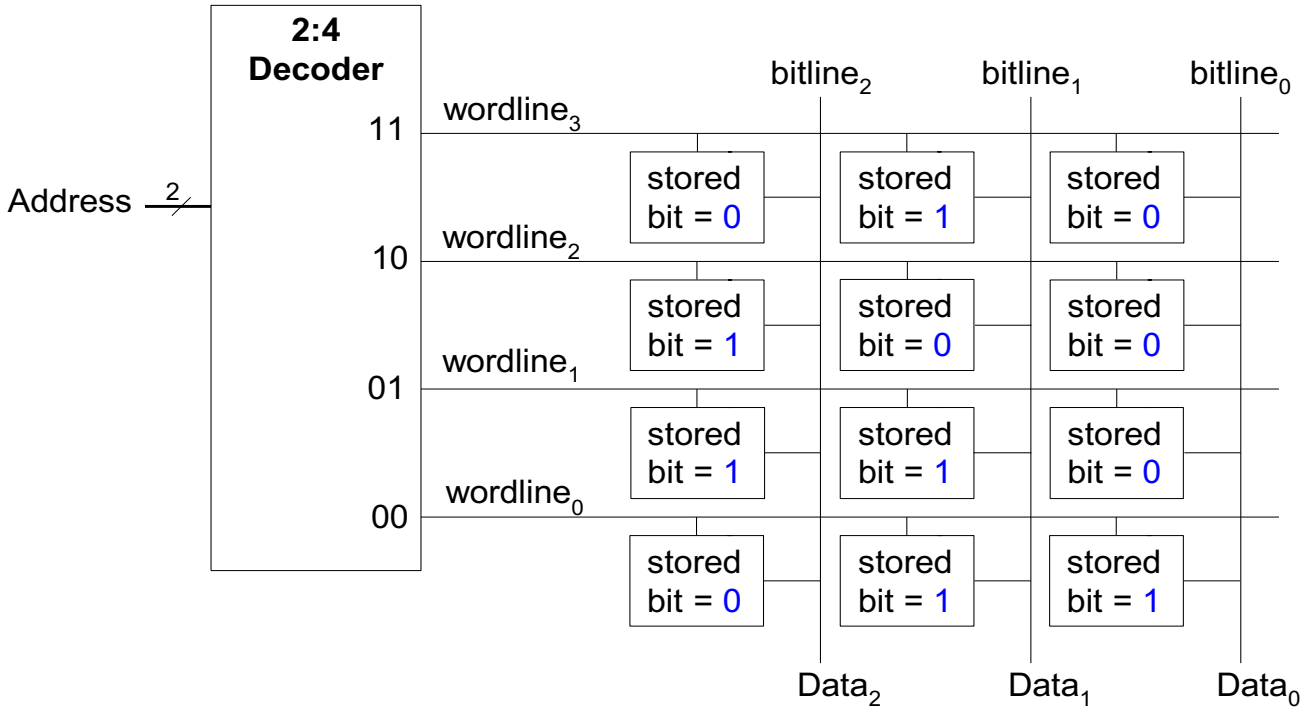
Example: Logic with ROMs

Implement the following logic functions using a $2^2 \times 3$-bit ROM:

- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$



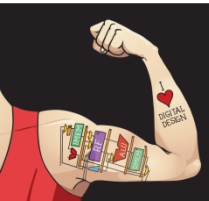
Logic with Any Memory Array



$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = \overline{A}_1 + A_0$$

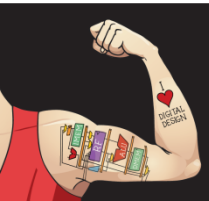
$$Data_0 = \overline{A}_1 \overline{A}_0$$



Logic with Memory Arrays

Implement the following logic functions using a $2^2 \times 3$-bit memory array:

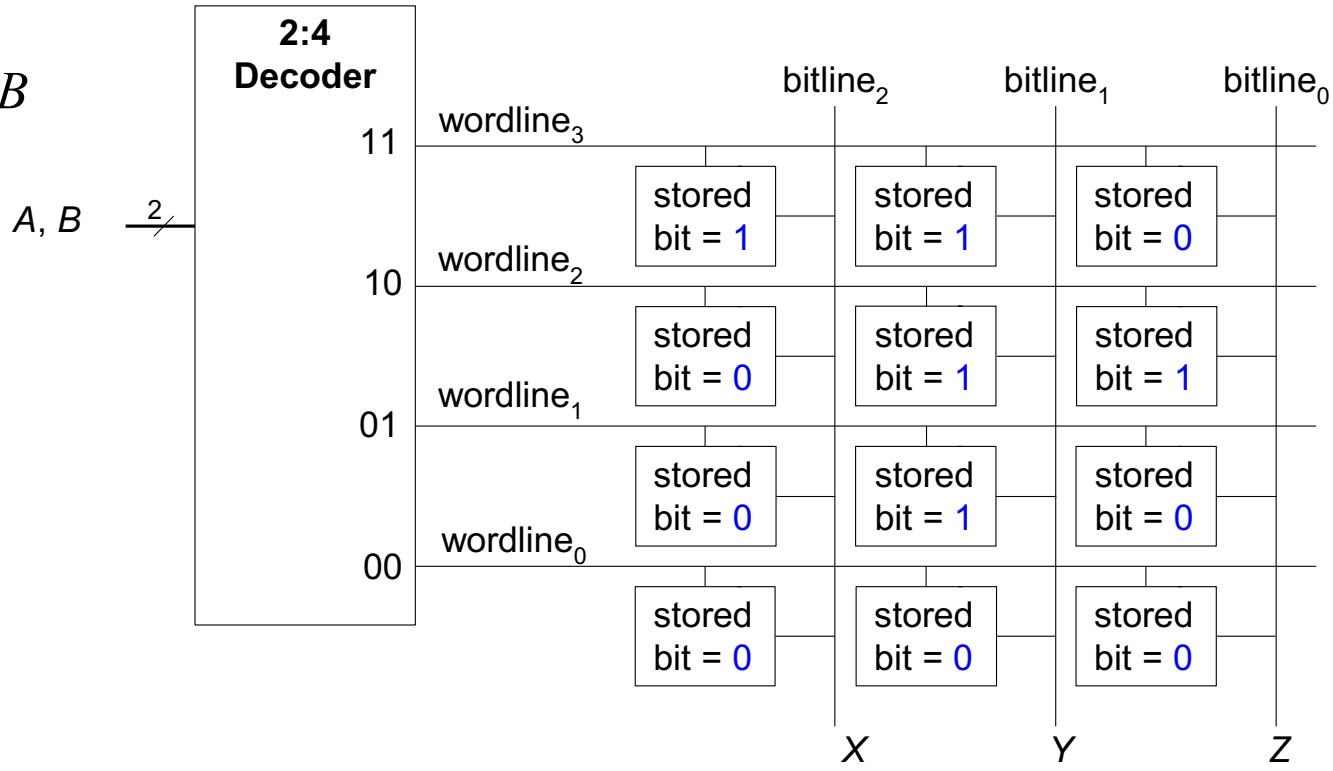
- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$

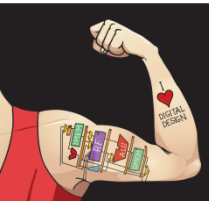


Logic with Memory Arrays

Implement the following logic functions using a $2^2 \times 3$-bit memory array:

- $X = AB$
- $Y = A + B$
- $Z = A\overline{B}$



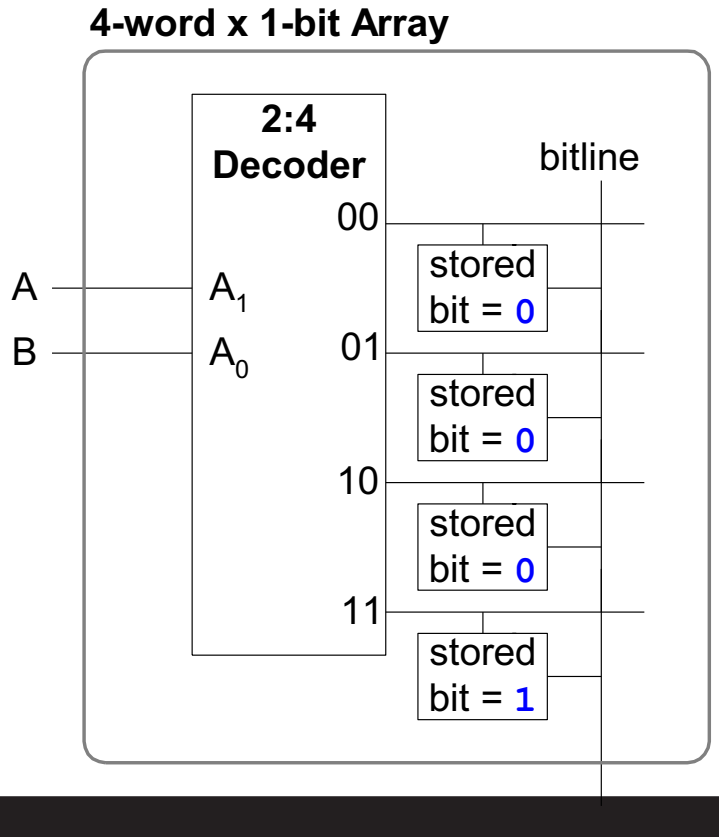


Logic with Memory Arrays

Called *lookup tables* (LUTs): look up output at each input combination (address)

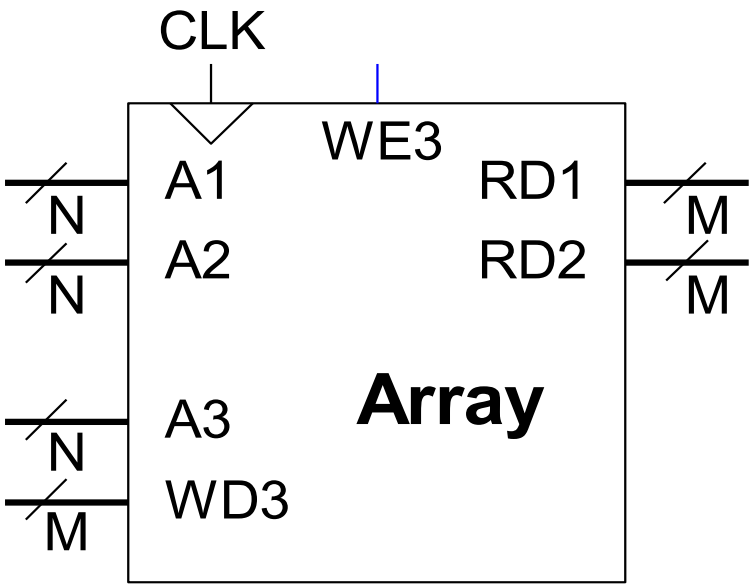
Truth Table

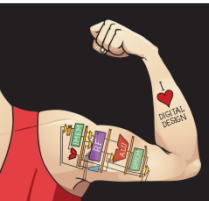
A	B	Y
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**



Multi-ported Memories

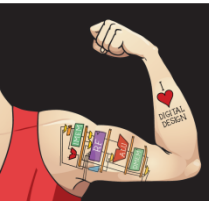
- **Port:** address/data pair
- 3-ported memory
  - 2 read ports (A1/RD1, A2/RD2)
  - 1 write port (A3/WD3, WE3 enables writing)
- **Register file:** small multi-ported memory





SystemVerilog Memory Arrays

```
// 256 x 64 memory module with one read/write port
module dmem(input  logic        clk, we,
            input  logic [7:0]  a,
            input  logic [63:0] wd,
            output logic [63:0] rd);

  logic  [63:0] RAM[255:0];

  always @(posedge clk)
    begin
      rd <= RAM[a];    // synchronous read
      if (we)
        RAM[a] <= wd; // synchronous write
    end
endmodule
```

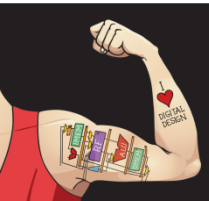


SystemVerilog Register File

```
// 16 x 32 register file with two read, 1 write port
module rf(input  logic        clk, we3,
          input  logic [3:0] a1, a2, a3,
          input  logic [31:0] wd3,
          output logic [31:0] rd1, rd2);

  logic  [31:0] RAM[15:0];

  always @(posedge clk)  // synchronous write
    if (we3)
        RAM[a3] <= wd3;
  assign rd1 = RAM[a1];  // asynchronous read
  assign rd2 = RAM[a2];
endmodule
```

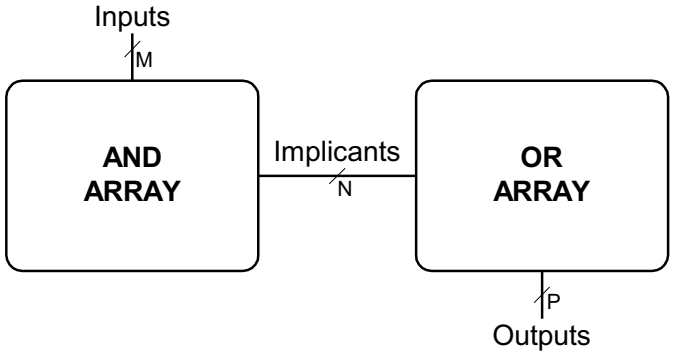
Logic Arrays

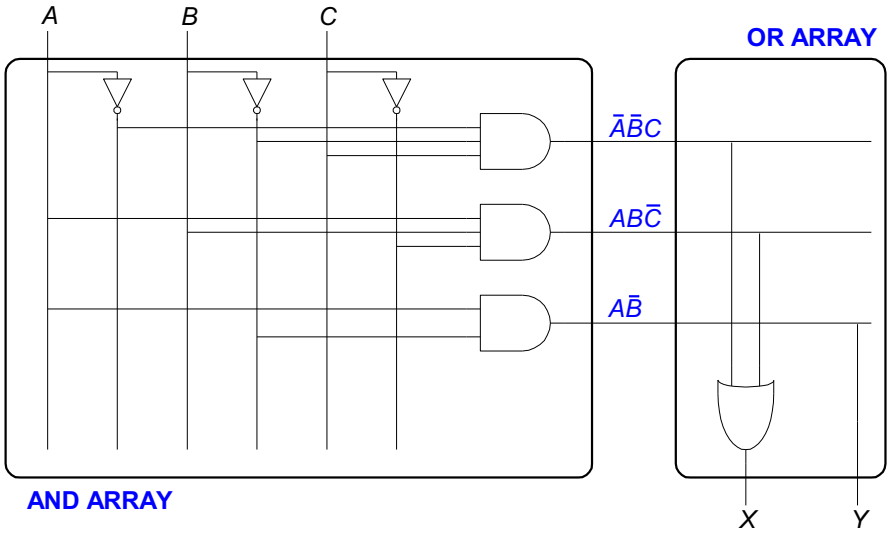
- **PLAs** (Programmable logic arrays)
  - AND array followed by OR array
  - Combinational logic only
  - Fixed internal connections
- **FPGAs** (Field programmable gate arrays)
  - Array of Logic Elements (LEs)
  - Combinational and sequential logic
  - Programmable internal connections



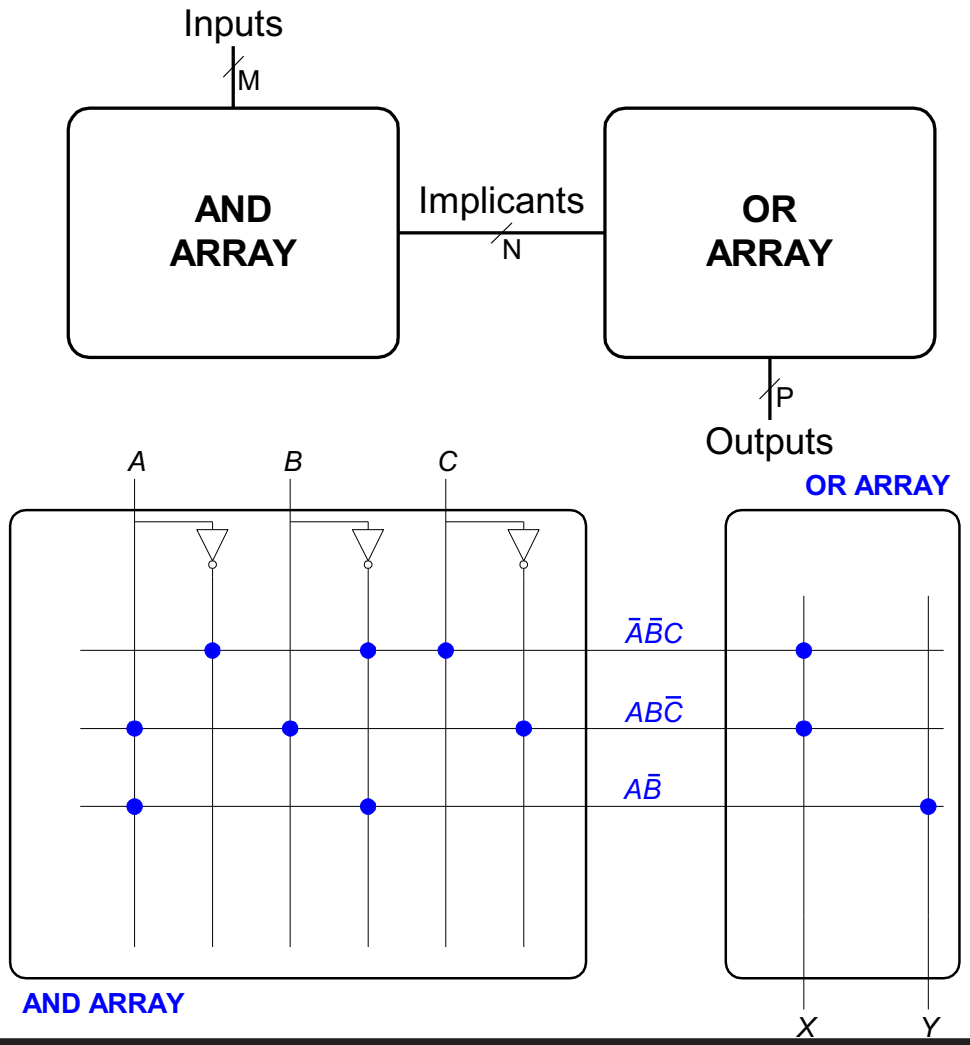
PLAs

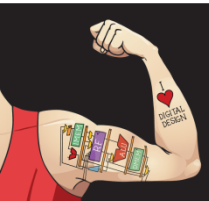
- $X = \overline{A}\,\overline{B}C + AB\overline{C}$
- $Y = A\overline{B}$





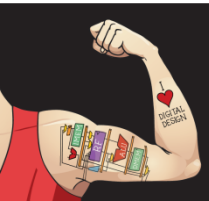
PLAs: Dot Notation



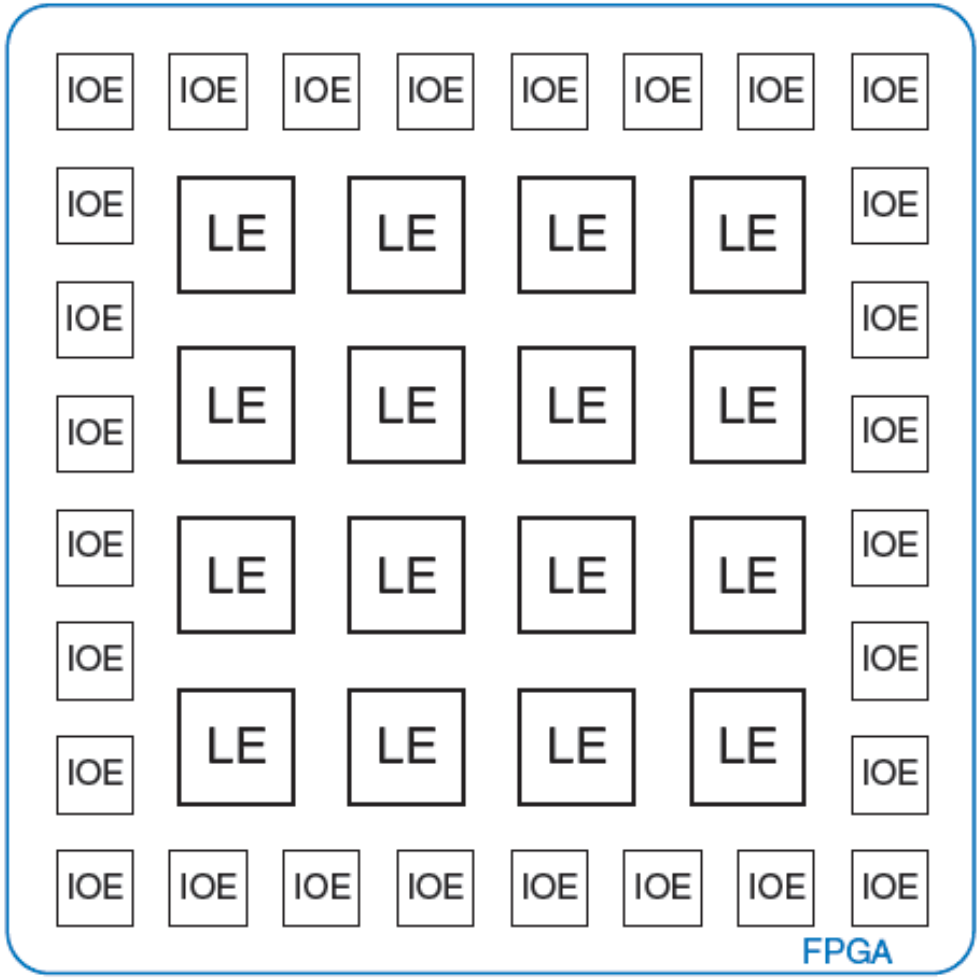


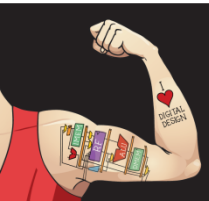
FPGA: Field Programmable Gate Array

- Composed of:
  - **LEs** (Logic elements): perform logic
  - **IOEs** (Input/output elements): interface with outside world
  - **Programmable interconnection:** connect LEs and IOEs
  - Some FPGAs include other building blocks such as multipliers and RAMs



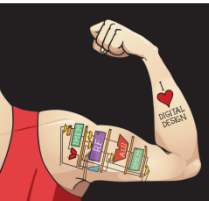
General FPGA Layout



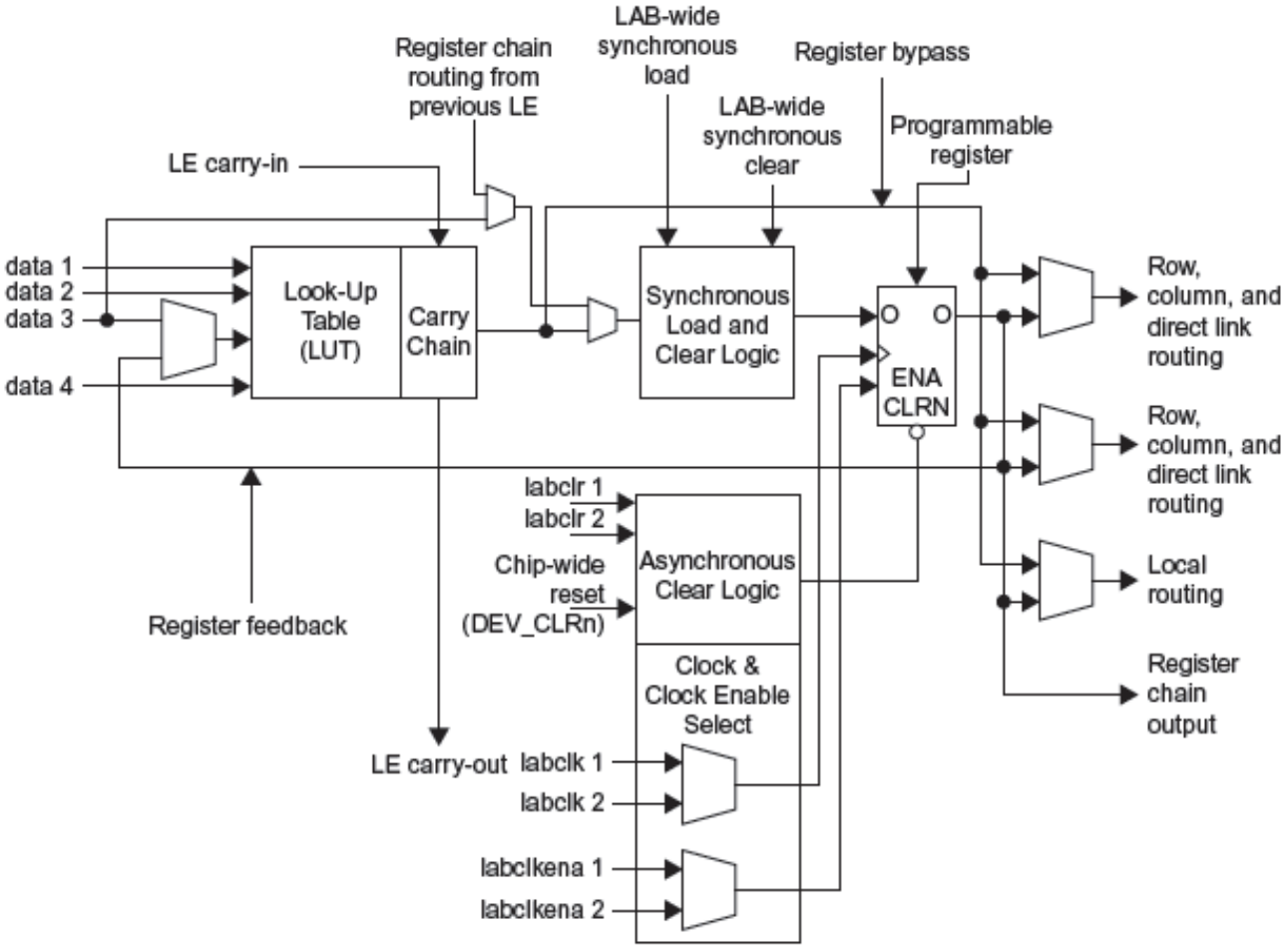


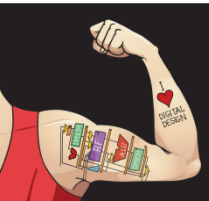
LE: Logic Element

- Composed of:
  - **LUTs** (lookup tables): perform combinational logic
  - **Flip-flops:** perform sequential logic
  - **Multiplexers:** connect LUTs and flip-flops



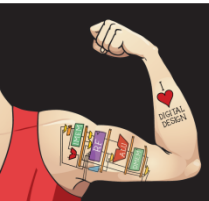
Altera Cyclone IV LE





Altera Cyclone IV LE

- The Altera Cyclone IV LE has:
  - 1 four-input LUT
  - 1 registered output
  - 1 combinational output



LE Configuration Example

Show how to configure a Cyclone IV LE to perform the following functions:

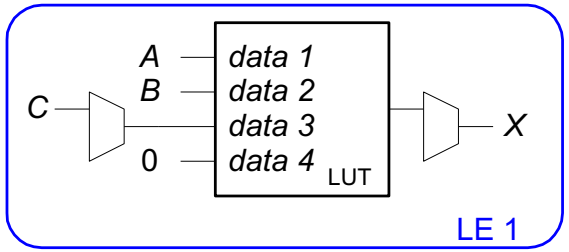
- $X = \overline{A}\,\overline{B}C + AB\overline{C}$
- $Y = A\overline{B}$

LE Configuration Example

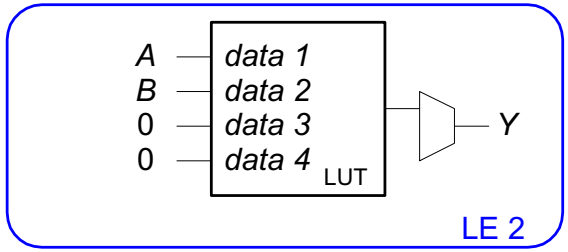
Show how to configure a Cyclone IV LE to perform the following functions:

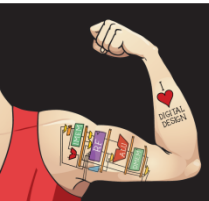
- $X = \overline{A}\,\overline{B}C + AB\overline{C}$
- $Y = A\overline{B}$

(A) data 1	(B) data 2	(C) data 3	data 4	(X) LUT output
0	0	0	X	0
0	0	1	X	1
0	1	0	X	0
0	1	1	X	0
1	0	0	X	0
1	0	1	X	0
1	1	0	X	1
1	1	1	X	0



(A) data 1	(B) data 2	data 3	data 4	(Y) LUT output
0	0	X	X	0
0	1	X	X	0
1	0	X	X	1
1	1	X	X	0



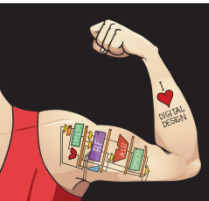


LE Example: AND5

How many LEs are required to build a 5-input AND gate?

LE Example: 8-bit shift register

How many LEs are required to build an 8-bit shift register?



LE Example: 3-bit counter

How many LEs are required to build a 3-bit counter?

FPGA Design Flow

Using a CAD tool (such as Altera's Quartus II)

- **Enter the design** with a HDL
- **Simulate** the design
- **Synthesize** design and map it onto FPGA
- **Download the configuration** onto the FPGA
- **Test** the design

This is an iterative process!