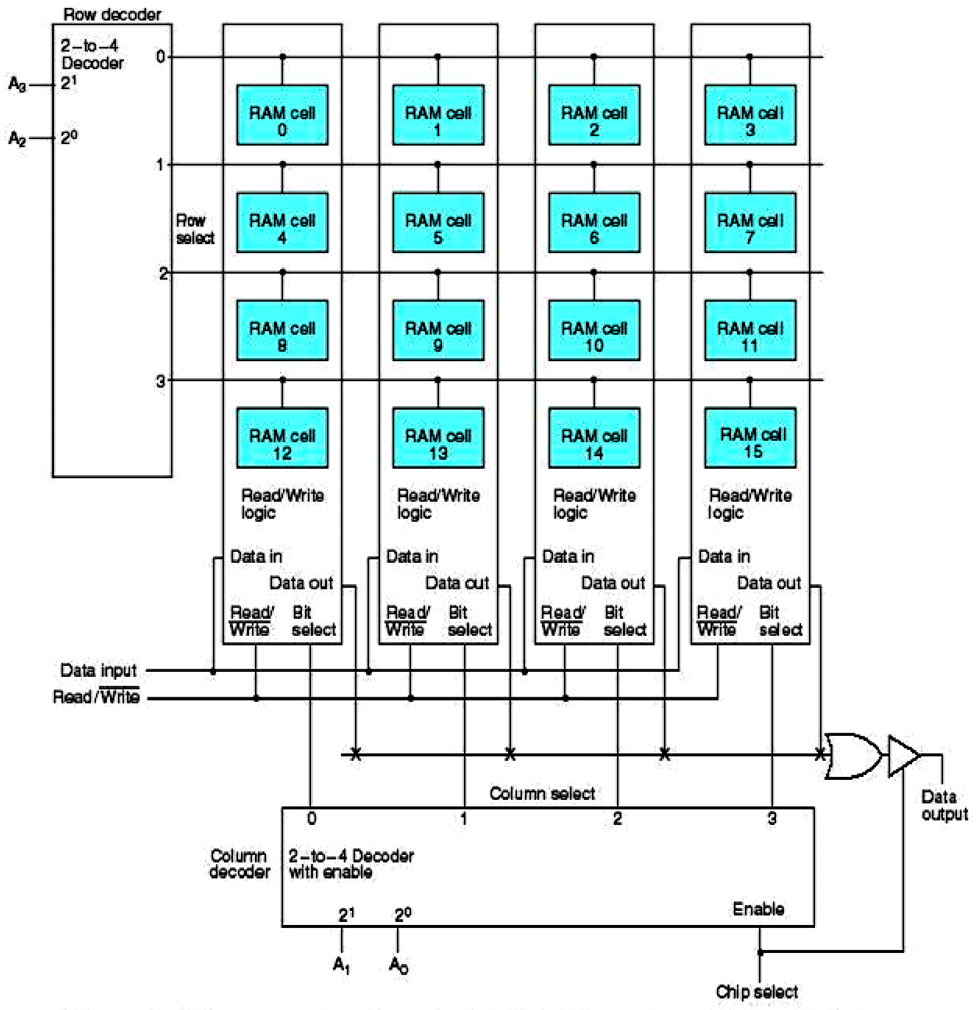
RAM and ROM

Instructor : Pei-Yun Tsai

RAM Cell Array



Write and Read Operations

- Write to RAM
  - Apply the binary address of the desired word to the address lines
  - Apply the data bits that must be stored in memory to the data input lines
  - Activate the write control
- Read from RAM
  - Apply the binary address of the desired word to the address lines
  - Activate the read control

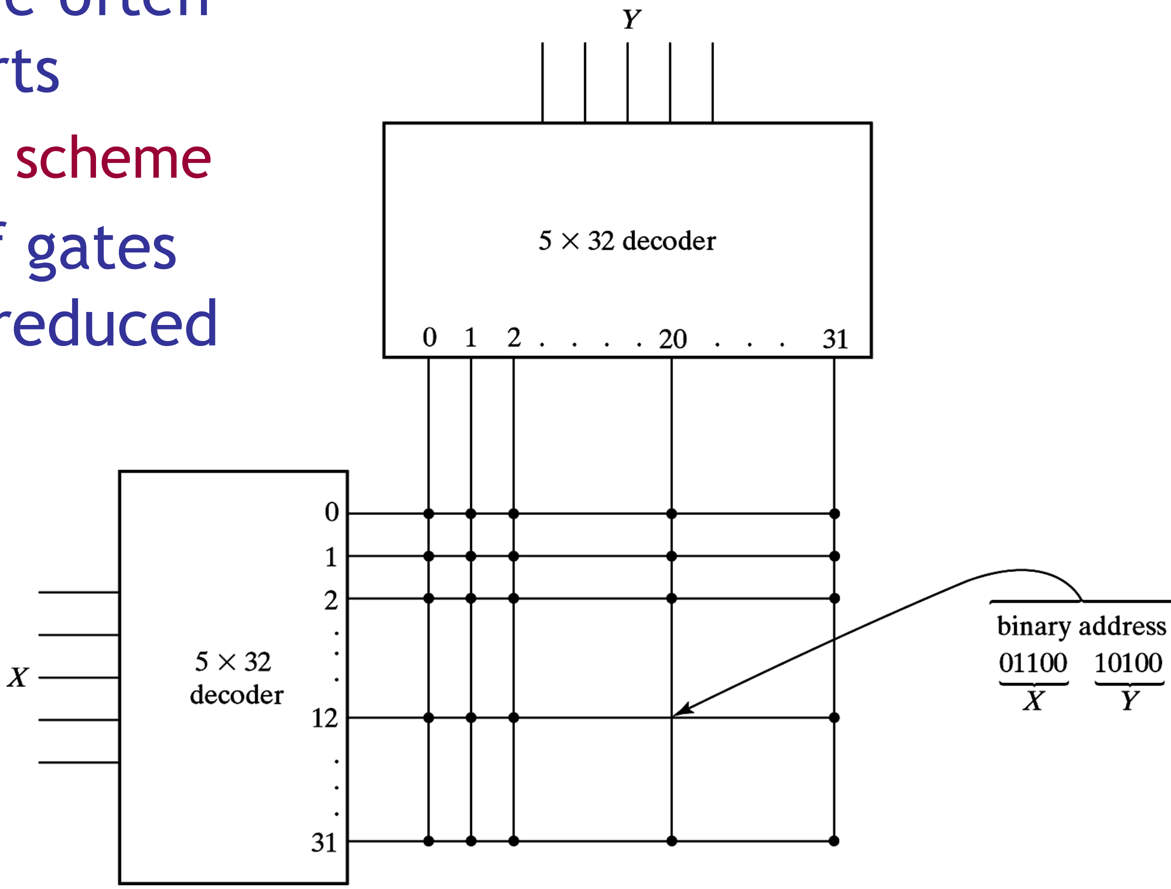
Coincident Decoding

- Address decoders are often divided into two parts
  - A two-dimensional scheme
- The total number of gates in decoders can be reduced
- Can arrange the memory cells to a square shape
- EX: 10-bit address

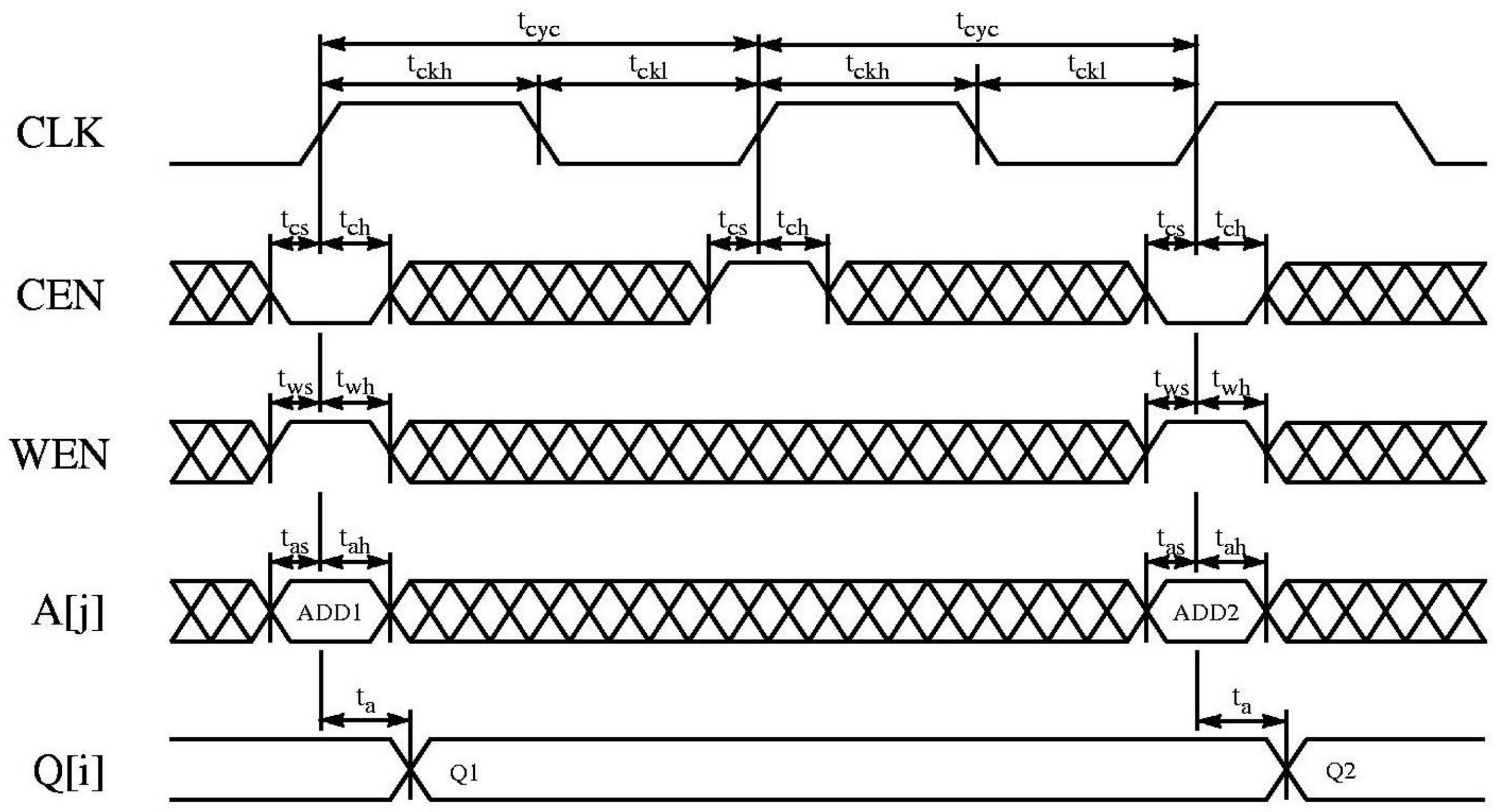
  404 = 0110010100

  X = 01100

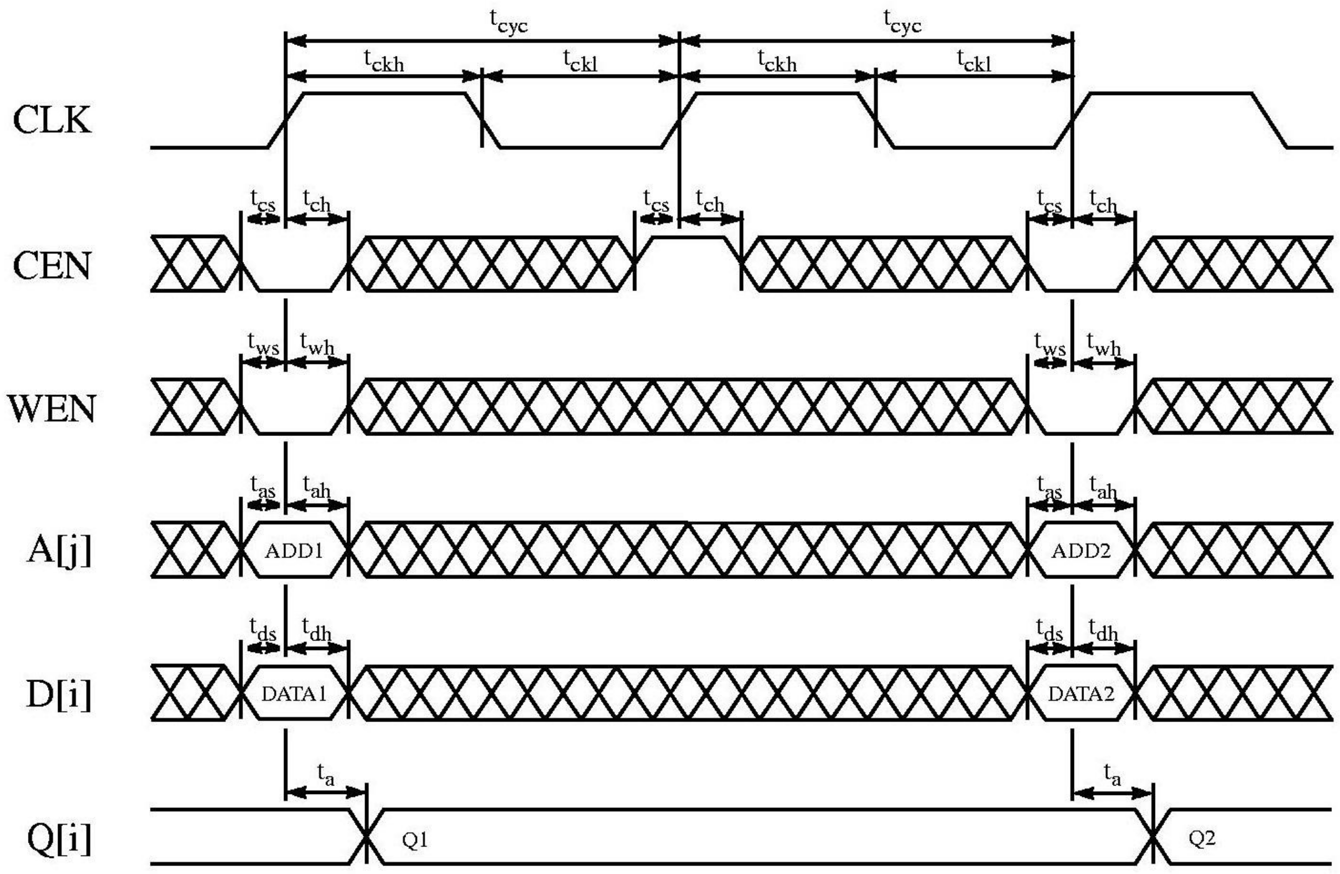
  Y = 10100



Timing of Synchronous Read Operation



Timing of Synchronous Write Operation



Modeling Memory Array (1/2)

- The following codes do not synthesize a real SRAM module. Instead, an array of registers is constructed. If a real SRAM module is required, you need to call SRAM compiler.

```
reg [3:0] MemCell [7:0]; // 8 4-bit memory array
reg [3:0] RAMOut;
wire[3:0] NewData;


//Asynchronous Memory array operation
always@(MemCell[7] or NewData)
begin
        //read
        RAMOut=MemCell[7];
        //write
        MemCell[0]=NewData;
end
```

Modeling Memory Array (2/2)

```
reg [3:0] MemCell [7:0]; // 8 4-bit memory array
reg [3:0] RAMOut;
wire[3:0] NewData;


//Synchronous Memory array operation
always@(posedege CLK)
begin
        //read
        RAMOut=MemCell[7];
        //write
        if (WEN)
        MemCell[0]=NewData;
end
```

Instantiation SRAM made by SRAM compiler

```
SRAM256x28
SRAMU0(.Q(dDataOut),
        .CLK(clk),
        .CEN(RAMCEN),
        .WEN(bRW),
        .A(dAdd),
        .D(dDataIn),
        .OEN(bOEN));
```

Memory Array Initialization

- Use memory array declaration in Verilog
  - Use in testbench
  - $readmemb, $readmemh

```
module RAMInit;
    reg [3:0] Memory [7:0]; // 8 4-bit memory array
    integer i;
    initial
       begin
         // read from file
            $readmemb("init.txt", Memory);
            for (i=0; i<8; i=i+1)
               $display("Memory[%d]=%b",i,Memory[i]);
       end
endmodule
```

init.txt

```
@02
0011 0101
1100 1010
@06
11zz 0000
```

Result

```
Memory[0]=xxxx
Memory[1]=xxxx
Memory[2]=0011
Memory[3]=0101
Memory[4]=1100
Memory[5]=1010
Memory[6]=11zz
Memory[7]=0000
```

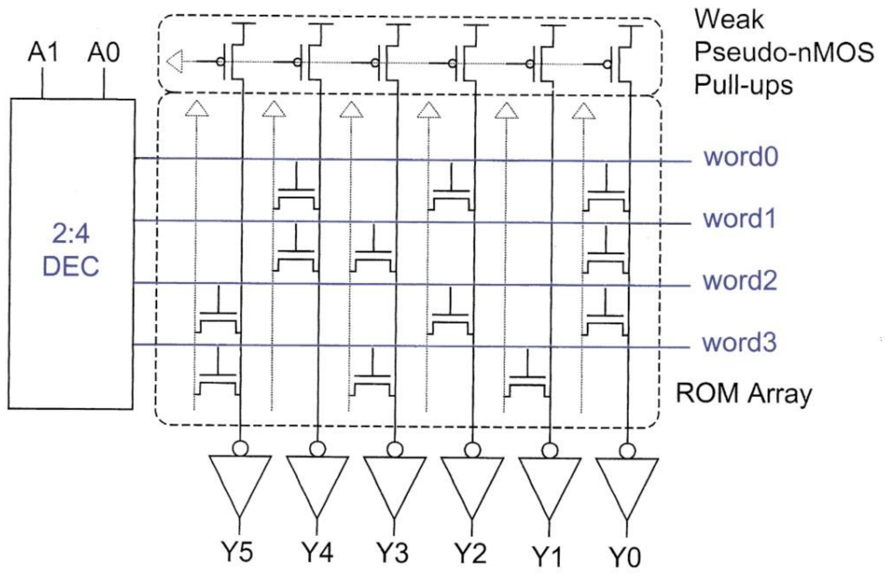
Read-Only Memories

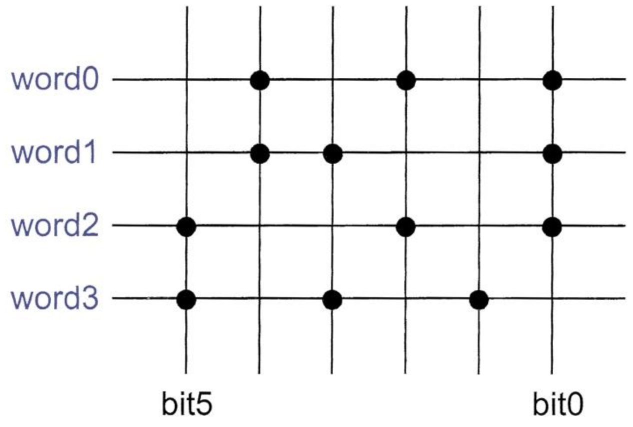
- Read-only memories are nonvolatile
  - Retain their contents when power is removed
- Mask-programmed ROMs use one transistor per bit
  - Presence or absence determines 1 or 0

ROM Example

- 4-word x 6-bit ROM
  - Represented with dot diagram
  - Dots indicate 1's in ROM

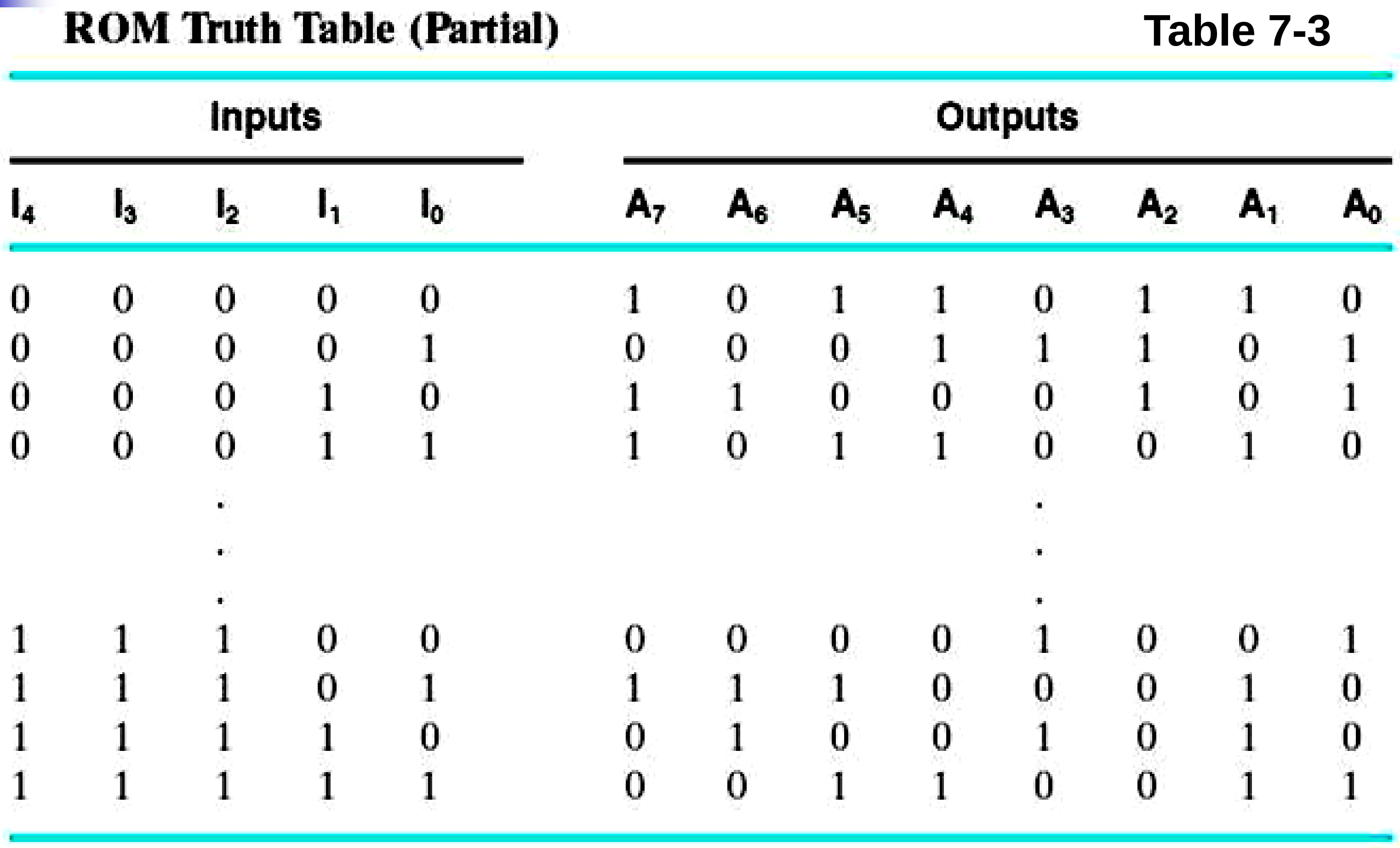
```
Word 0: 010101
Word 1: 011001
Word 2: 100101
Word 3: 101010
```





Read Only Memory (1/2)

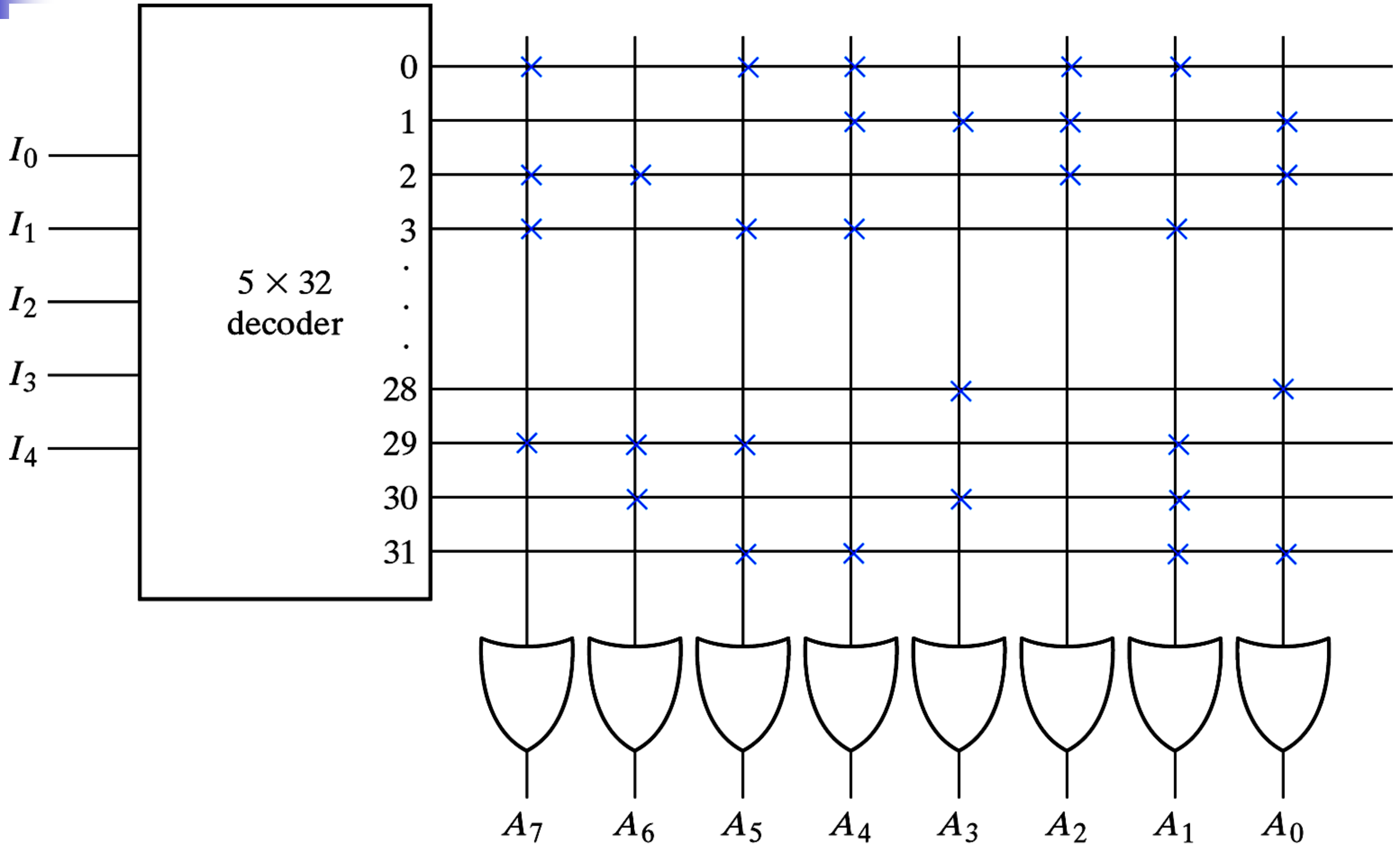
Table 7-3



ROM Truth Table (Partial)

Inputs					Outputs							
I_4	I_3	I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	1	0	1	1	0	1	1	0
0	0	0	0	1	0	0	0	1	1	1	0	1
0	0	0	1	0	1	1	0	0	0	1	0	1
0	0	0	1	1	1	0	1	1	0	0	1	0
		.							.			
		.							.			
		.							.			
1	1	1	0	0	0	0	0	0	1	0	0	1
1	1	1	0	1	1	1	1	0	0	0	1	0
1	1	1	1	0	0	1	0	0	1	0	1	0
1	1	1	1	1	0	0	1	1	0	0	1	1

Read Only Memory (2/2)



Modeling ROM (1/2)

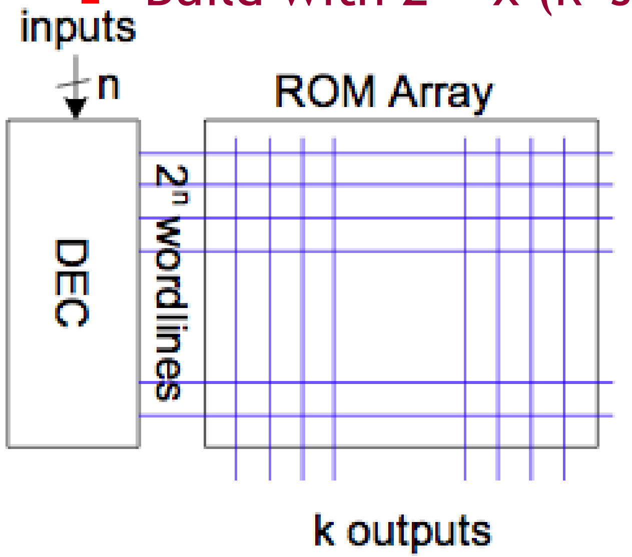
```
module rom_256x8a (OE, Addr, O);
 parameter numAddr = 8;     // word depth = (2^numAddr);
 parameter numOut  = 8;     // number of Output bit width
 input               OE  ; // output enable
 input  [numAddr-1:0] Addr; // input address
 output [numOut-1:0]  O   ; // output data
 reg [numOut-1:0]     DATA; // ROM Data

 always @(Addr)     begin      // In this exapmle, it just use the 225 entries.
   case (Addr)             //Address      : Data
   //===================================
   8'b00000000 : DATA = 8'b11111111;
   8'b00000001 : DATA = 8'b11111111;
   8'b00000010 : DATA = 8'b11111111;
   8'b00000011 : DATA = 8'b11111111;
   8'b00000100 : DATA = 8'b11111111;
   8'b00000101 : DATA = 8'b11111111;
   8'b00000110 : DATA = 8'b11111111;
   ....
```

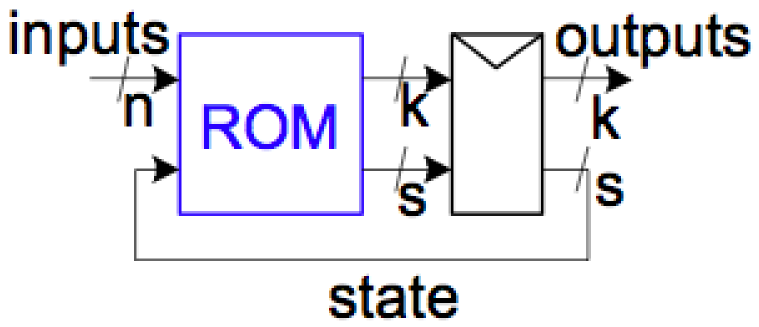
Modeling ROM (2/2)

```
    8'b11011111 : DATA = 8'b01111001;
    8'b11100000 : DATA = 8'b01000111;
    default     : DATA = 8'bxxxxxxxx;
    //===================================
    endcase
    end
    // ---------------------------------------------------
    // OE low to output in Hi-Z
    // OE high to output driven
    // ---------------------------------------------------
    assign O = (OE)? DATA : 8'bz;  // not mandatory
endmodule
```

Building Logic with ROMs

- Use ROM as lookup table containing truth table
  - n inputs, k outputs requires 2^n words x k bits
  - Changing function is easy - reprogram ROM
- Finite State Machine
  - n inputs, k outputs, s bits of state
  - Build with 2^{n+s} x (k+s) bit ROM and (k+s) bit reg


inputs
n
ROM Array
DEC
2^n wordlines
k outputs


inputs
n
ROM
k
s
outputs
k
s
state