



## Memories & More

- Memories in Verilog
- Memories on the FPGA
- External Memories
  - SRAM (async, sync)
  - DRAM
  - Flash

## Memories: a practical primer

- The good news: huge selection of technologies
  - Small & faster vs. large & slower
  - Every year capacities go up and prices go down
  - New kid on the block: high density, fast flash memories
    - Non-volatile, read/write, no moving parts! (robust, efficient)
- The bad news: perennial system bottleneck
  - Latencies (access time) haven't kept pace with cycle times
  - Separate technology from logic, so must communicate between silicon, so physical limitations (# of pins, R's and C's and L's) limit bandwidths
    - New hopes: capacitive interconnect, 3D IC's
  - Likely the limiting factor in cost & performance of many digital systems: designers spend a lot of time figuring out how to keep memories running at peak bandwidth
  - "It's the memory - just add more faster memory"

## Memories in Verilog

- `reg bit; // a single register`
- `reg [31:0] word; // a 32-bit register`
- `reg [31:0] array[15:0]; // 16 32-bit regs`
- `reg [31:0] array_2d[31:0][15:0];`  
`// 2 dimensional 32-bit array`
- `wire [31:0] read_data, write_data;`  
`wire [3:0] index;`  
  
`// combinational (asynch) read`  
`assign read_data = array[index];`  
  
`// clocked (synchronous) write`  
`always @(posedge clock)`  
`array[index] <= write_data;`

## Multi-port Memories (aka regfiles)

```
reg [31:0] regfile[30:0]; // 31 32-bit words

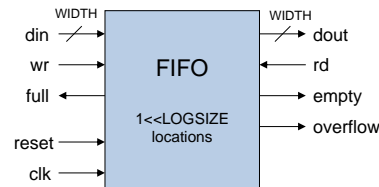
// Beta register file: 2 read ports, 1 write
wire [4:0] ra1, ra2, wa;
wire [31:0] rd1, rd2, wd;

assign ra1 = inst[20:16];
assign ra2 = ra2sel ? inst[25:21] : inst[15:11];
assign wa = wasel ? 5'd30 : inst[25:21];

// read ports
assign rd1 = (ra1 == 5'd31) ? 32'd0 : regfile[ra1];
assign rd2 = (ra2 == 5'd31) ? 32'd0 : regfile[ra2];
// write port
always @(posedge clk)
  if (werf) regfile[wa] <= wd;

assign z = ~| rd1; // used in BEQ/BNE instructions
```

## FIFOs

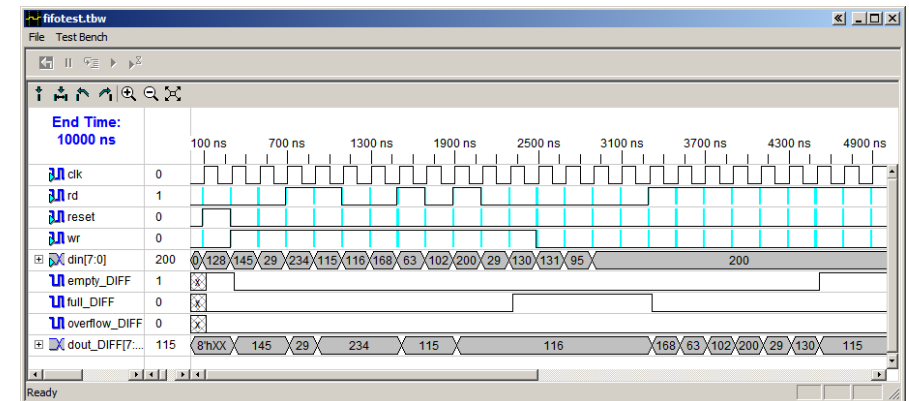


```
// a simple synchronous FIFO (first-in first-out) buffer
// Parameters:
// LOGSIZE      (parameter) FIFO has 1<<LOGSIZE elements
// WIDTH        (parameter) each element has WIDTH bits
// Ports:
// clk          (input) all actions triggered on rising edge
// reset        (input) synchronously empties fifo
// din          (input, WIDTH bits) data to be stored
// wr           (input) when asserted, store new data
// full         (output) asserted when FIFO is full
// dout         (output, WIDTH bits) data read from FIFO
// rd           (input) when asserted, removes first element
// empty        (output) asserted when fifo is empty
// overflow     (output) asserted when wr but no room, cleared on next RD
```

```
module fifo #(parameter LOGSIZE = 2,    // default size is 4 elements
               WIDTH = 4)              // default width is 4 bits
(input clk,reset,wr,rd, input [WIDTH-1:0] din,
 output full,empty,overflow, output [WIDTH-1:0] dout);
...
endmodule
```

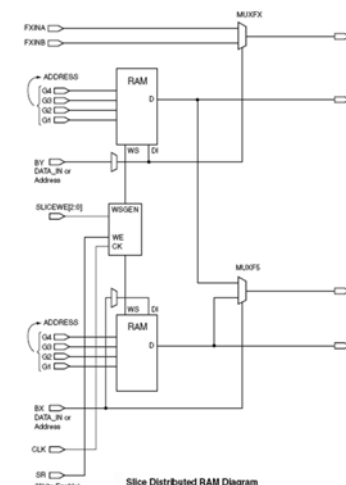
## FIFOs in action

```
// make a fifo with 8 8-bit locations
fifo f8x8 #(C.LOGSIZE(3),.WIDTH(8))
(.clk(clk),.reset(reset),
 .wr(wr),.din(din),.full(full),
 .rd(rd),.dout(dout),.empty(empty),
 .overflow(overflow));
```

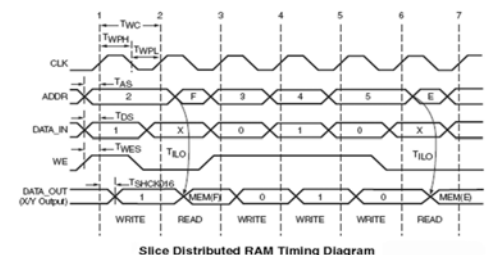


## FPGA memory implementation

- Regular registers in logic blocks
  - Piggy use of resources, but convenient & fast if small
- [Xilinx Vertex II] use the LUTs:
  - Single port:  $16 \times (1, 2, 4, 8)$ ,  $32 \times (1, 2, 4, 8)$ ,  $64 \times (1, 2)$ ,  $128 \times 1$
  - Dual port (1 R/W, 1R):  $16 \times 1$ ,  $32 \times 1$ ,  $64 \times 1$
  - Can fake extra read ports by cloning memory: all clones are written with the same addr/data, but each clone can have a different read address
- [Xilinx Vertex II] use block ram:
  - 18K bits:  $16K \times 1$ ,  $8K \times 2$ ,  $4K \times 4$   
with parity:  $2K \times (8+1)$ ,  $1K \times (16+2)$ ,  $512 \times (32+4)$
  - Single or dual port
  - Pipelined (clocked) operations
  - Labkit XCV2V6000: 144 BRAMs, 2952K bits total



## LUT-based RAMs



CLB Distributed RAM Switching Characteristics						
Description		Symbol	Speed Grade			Units
			-6	-5	-4	
Sequential Delays						
Clock CLK to X/Y outputs (WE active) in 16 x 1 mode		T <sub>SHK0H16</sub>	1.63	1.79	2.05	ns, Max
Clock CLK to X/Y outputs (WE active) in 32 x 1 mode		T <sub>SHK0H32</sub>	1.97	2.17	2.49	ns, Max
Clock CLK to F5 output		T <sub>SHK0F5</sub>	1.77	1.94	2.23	ns, Max
Setup and Hold Times Before/After Clock CLK						
8K/8Y data inputs (D/E)		T <sub>CD/TCH</sub>	0.53/-0.09	0.58/-0.10	0.67/-0.11	ns, Min
F/G address inputs		T <sub>AD/TAH</sub>	0.40/-0.00	0.44/-0.00	0.50/-0.00	ns, Min
SR input (WS)		T <sub>CD/TCH</sub>	0.42/-0.01	0.46/-0.01	0.53/-0.01	ns, Min
Clock CLK						
Minimum Pulse Width, High		T <sub>WH</sub>	0.57	0.63	0.72	ns, Min
Minimum Pulse Width, Low		T <sub>WL</sub>	0.57	0.63	0.72	ns, Min
Minimum clock period to meet address/write cycle time		T <sub>WC</sub>	1.14	1.25	1.44	ns, Min
Combinatorial Delays						
4-input function, F/G inputs to X/Y outputs		T <sub>4F</sub>	0.25	0.39	0.44	ns, Max

## LUT-based RAM Modules

## Tools will often build these for you...

From Lab 2:

```
reg [7:0] segments;
always @ (switch[3:0]) begin
    case (switch[3:0])
        4'h0: segments[6:0] = 7'b0111111;
        4'h1: segments[6:0] = 7'b0000110;
        4'h2: segments[6:0] = 7'b0101011;
        4'h3: segments[6:0] = 7'b1001111;
        4'h4: segments[6:0] = 7'b1100110;
        4'h5: segments[6:0] = 7'b1101101;
        4'h6: segments[6:0] = 7'b1111101;
        4'h7: segments[6:0] = 7'b0000111;
        4'h8: segments[6:0] = 7'b1111111;
        4'h9: segments[6:0] = 7'b1100111;
        4'hA: segments[6:0] = 7'b1110111;
        4'hB: segments[6:0] = 7'b1111100;
        4'hC: segments[6:0] = 7'b1011000;
        4'hD: segments[6:0] = 7'b1011110;
        4'hE: segments[6:0] = 7'b1111001;
        4'hF: segments[6:0] = 7'b1110001;
    default: segments[6:0] = 7'b00000000;
    endcase
    segments[7] = 1'b0; // decimal point
end
```

```
=====
*          HDL Synthesis          *
=====
Synthesizing Unit <lab2_2>.
  Related source file is "../lab2_2.v".
  ...
  Found 16x7-bit ROM for signal <$n0000>.
  ...
  Summary:
    inferred 1 ROM(s).
  ...
Unit <lab2_2> synthesized.

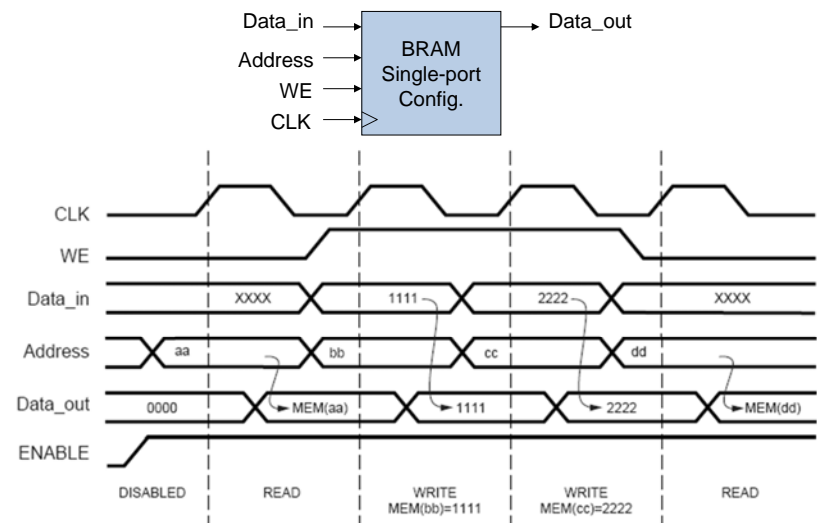
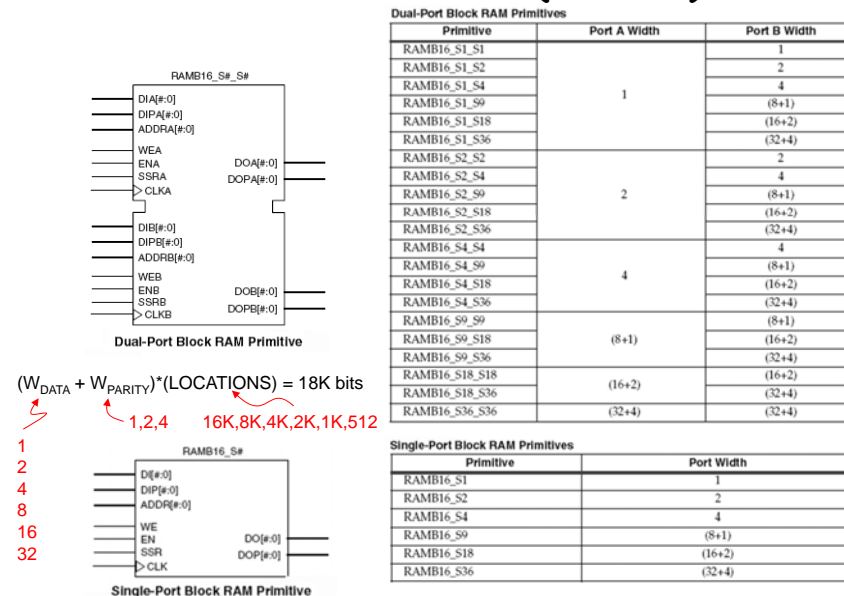
=====
Timing constraint: Default path analysis
Total number of paths / destination ports: 28 / 7
-----
Delay: 7.244ns (Levels of Logic = 3)
Source: switch<3> (PAD)
Destination: user1<0> (PAD)

Data Path: switch<3> to user1<0>
-----
Cell:in->out fanout Delay Delay Logical Name
-----
IBUF:I->O 7 0.825 1.102 switch_3_IBUF
LUT4:I0->O 1 0.439 0.517 Mrom_n0000_inst_lut4_01
OBUF:I->O 4.361 user1_0_OBUF
-----
Total 7.244ns (5.625ns logic, 1.619ns route)
(77.7% logic, 22.3% route)
```

```
// instantiate a LUT-based RAM module
RAM16X1S mymem #(.INIT(16'b0110_1111_0011_0101_1100)) // msb first
(.D(din),.O(dout),.WE(we),.WCLK(clock_27mhz),
.A0(a[0]),.A1(a[1]),.A2(a[2]),.A3(a[3]));
```

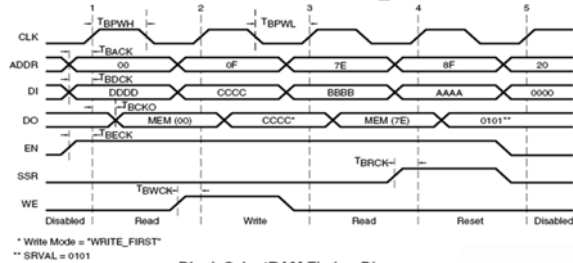
## Block Memories (BRAMs)

## BRAM Operation



Source: Xilinx App Note 463

## BRAM timing



Block SelectRAM Timing Diagram

Block SelectRAM Switching Characteristics

Description	Symbol	Speed Grade			Units
		-6	-5	-4	
Sequential Delays					
Clock CLK to DOUT output	T <sub>BCKO</sub>	2.10	2.31	2.65	ns, Max
Setup and Hold Times Before Clock CLK					
ADDR inputs	T <sub>BACK</sub> /T <sub>BCKA</sub>	0.29/ 0.00	0.32/ 0.00	0.36/ 0.00	ns, Min
DIN inputs	T <sub>BCKO</sub> /T <sub>BCKD</sub>	0.29/ 0.00	0.32/ 0.00	0.36/ 0.00	ns, Min
EN input	T <sub>BECK</sub> /T <sub>BEKE</sub>	0.95/-0.46	1.04/-0.50	1.20/-0.58	ns, Min
RST input	T <sub>BRCK</sub> /T <sub>BRKR</sub>	1.31/-0.71	1.44/-0.78	1.65/-0.90	ns, Min
WEN input	T <sub>BWCK</sub> /T <sub>BCKW</sub>	0.57/-0.19	0.63/-0.21	0.72/-0.25	ns, Min
Clock CLK					
CLKA to CLKB setup time for different ports	T <sub>BCCS</sub>	1.0	1.0	1.0	ns, min
Minimum Pulse Width, High	T <sub>BPWH</sub>	1.17	1.29	1.48	ns, Min
Minimum Pulse Width, Low	T <sub>BPWL</sub>	1.17	1.29	1.48	ns, Min

6.111 Fall 2016

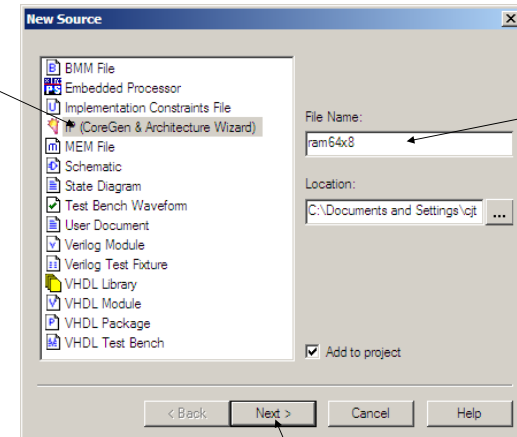
Lecture 12

13

## Using BRAMs (eg, a 64Kx8 ram)

- From menus: Project → New Source...

Select "IP"



Fill in name

Click "Next" when done...

6.111 Fall 2016

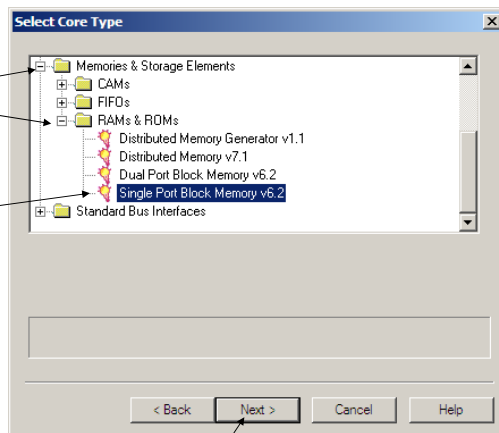
Lecture 12

14

## BRAM Example

Click open folders

Select "Single Port Block Memory"



Click "Next" and then "Finish" on next window

6.111 Fall 2016

Lecture 12

15

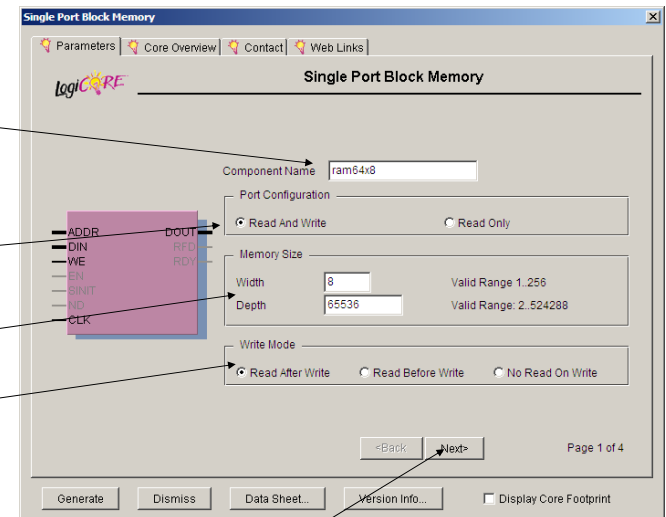
## BRAM Example

Fill in name (again?)

Select RAM vs ROM

Fill in width & depth

Usually "Read After Write" is what you want



Click "Next" ...

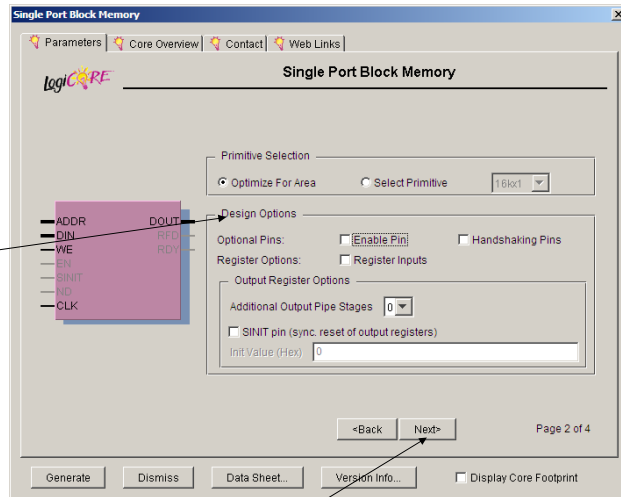
6.111 Fall 2016

Lecture 12

16

## BRAM Example

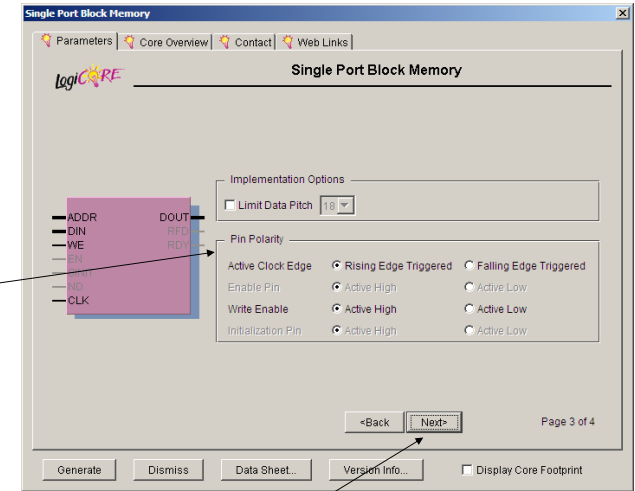
Can add extra control pins, but usually not



Click "Next" ...

## BRAM Example

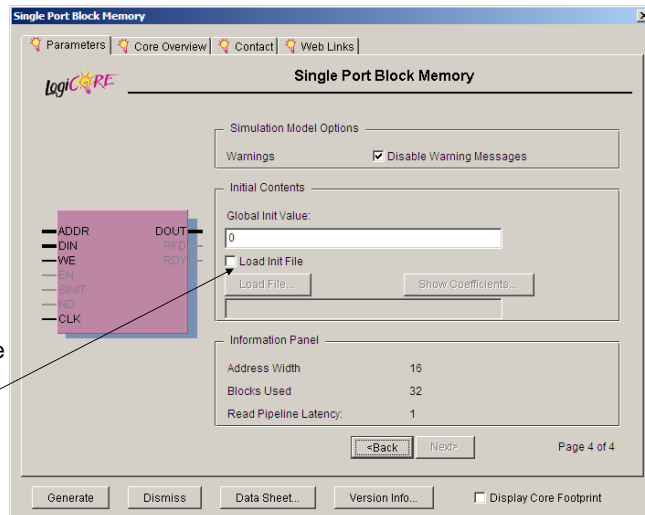
Select polarity of control pins; active high default is usually just fine



Click "Next" ...

## BRAM Example

Click to name a .coe file that specifies initial contents (eg, for a ROM)



Click "Generate" to complete

## .coe file format

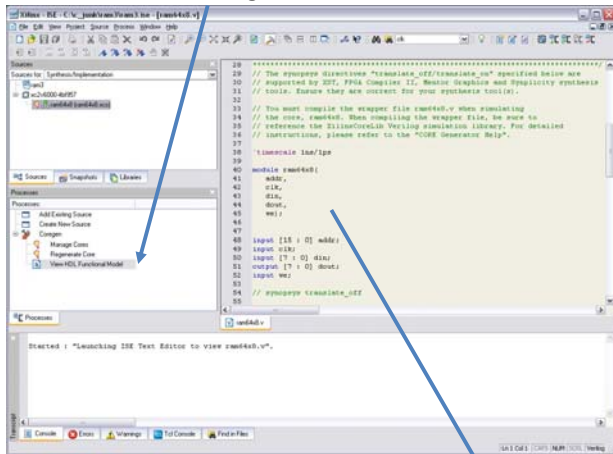
```
memory_initialization_radix=2;
memory_initialization_vector=
```

```
00000000,
00111110,
01100011,
00000011,
00000011,
00011110,
00000011,
00000011,
01100011,
00111110,
00000000,
00000000,
```

Memory contents with location 0 first, then location 1, etc. You can specify input radix, in this example we're using binary. MSB is on the left, LSB on the right. Unspecified locations (if memory has more locations than given in .coe file) are set to 0.

## Using result in your Verilog

- Look at generated Verilog for module definition (click on "View HDL Functional Model" under Coregen):



- Use to instantiate instances in your code:  
`ram64x8 foo(.addr(addr), .clk(clk), .we(we), .din(din), .dout(dout));`

## Memory Classification & Metrics

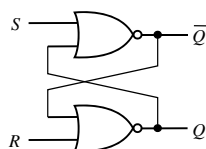
Read-Write Memory		Non-Volatile Read-Write Memory	Read-Only Memory
Random Access	Sequential Access		
SRAM DRAM	FIFO	EPROM E <sup>2</sup> PROM FLASH	Mask-Programmed ROM

### Key Design Metrics:

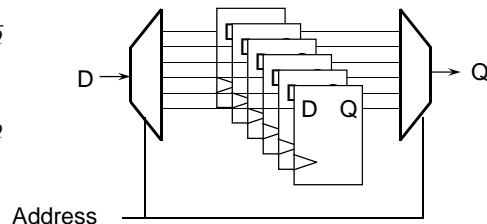
- Memory Density (number of bits/mm<sup>2</sup>) and Size
- Access Time (time to read or write) and Throughput
- Power Dissipation

## Static RAMs: Latch Based Memory

### Set Reset Flip Flop



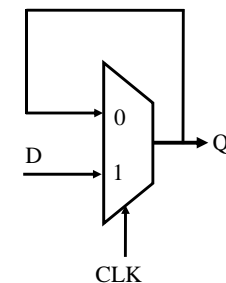
### Register Memory



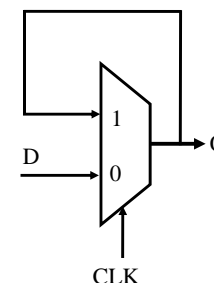
- Works fine for small memory blocks (e.g., small register files)
- Inefficient in area for large memories
- Density is the key metric in large memory circuits

## Latch and Register Based Memory

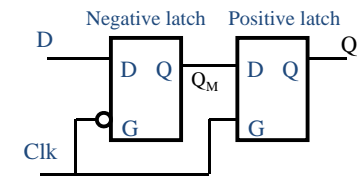
### Positive Latch



### Negative Latch

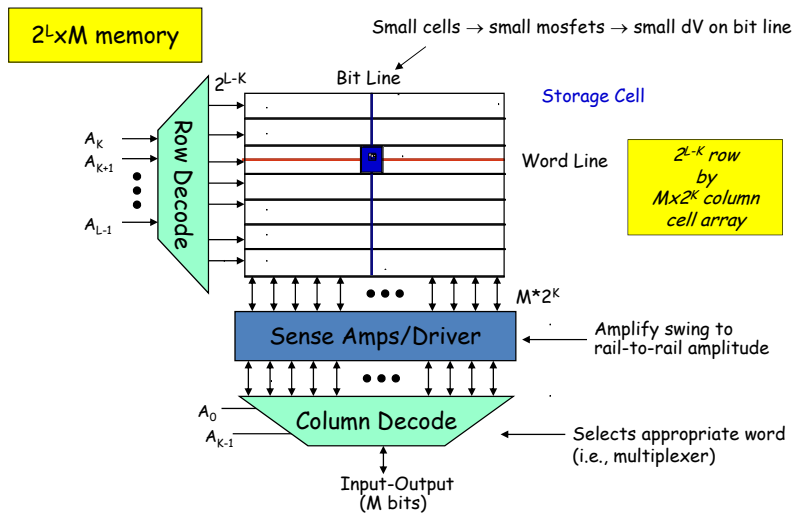


### Register Memory

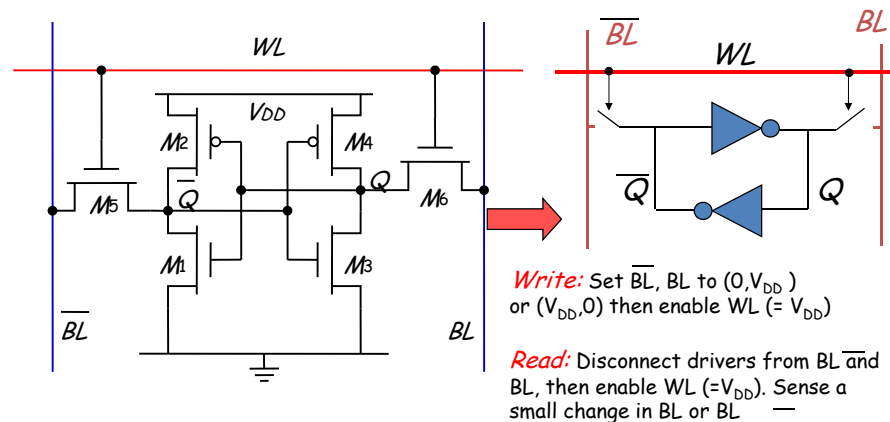


- Alternative view

## Memory Array Architecture

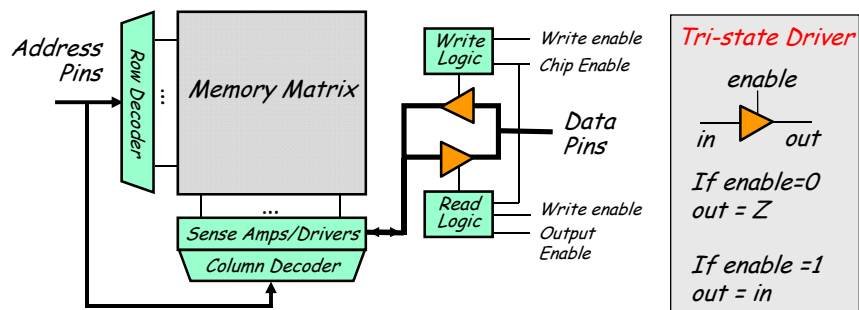


## Static RAM (SRAM) Cell (The 6-T Cell)



- State held by cross-coupled inverters (M1-M4)
- Retains state as long as power supply turned on
- Feedback must be overdriven to write into the memory

## Using External Memory Devices



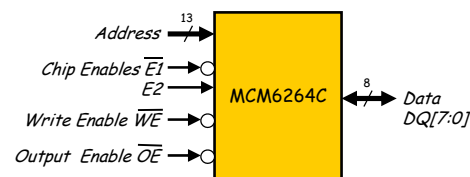
- Address pins drive row and column decoders
- Data pins are **bidirectional**: shared by reads and writes

- Output Enable gates the chip's tristate driver
- Write Enable sets the memory's read/write mode
- Chip Enable/Chip Select acts as a "master switch"

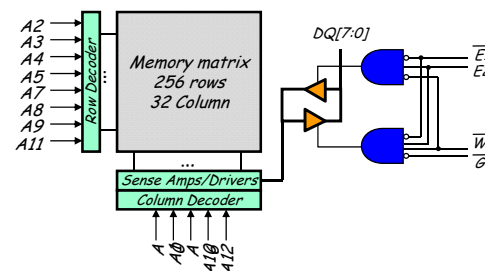
Concept of "Data Bus"

## MCM6264C 8K x 8 Static RAM

On the outside:



On the inside:



Same (bidirectional) data bus used for reading and writing

Chip Enables ( $\overline{E1}$  and  $\overline{E2}$ )

$\overline{E1}$  must be low and  $\overline{E2}$  must be high to enable the chip

Write Enable ( $\overline{WE}$ )

When low (and chip enabled), values on data bus are written to location selected by address bus

Output Enable ( $\overline{OE}$  or  $\overline{G}$ )

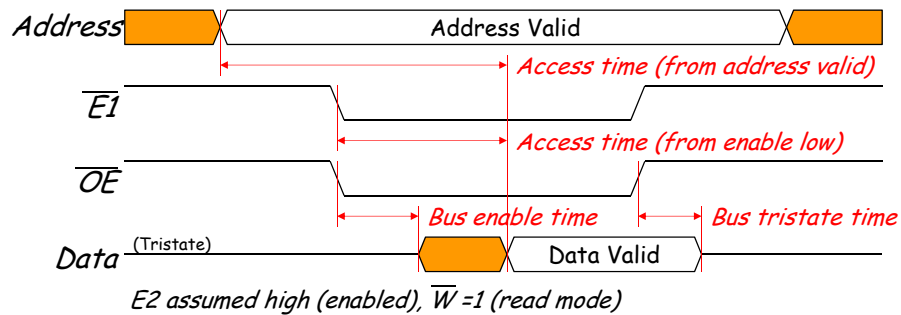
When low (and chip is enabled), data bus is driven with value of selected memory location

Pinout

NC	1	28	V <sub>CC</sub>
A17	2	27	$\overline{WE}$
A7	3	26	$\overline{E2}$
A6	4	25	A8
A5	5	24	A9
A4	6	23	A11
A3	7	22	$\overline{E1}$
A2	8	21	A10
A1	9	20	$\overline{OE}$
A0	10	19	DQ0
DQ0	11	18	DQ6
DQ1	12	17	DQ5
DQ2	13	16	DQ4
V <sub>SS</sub>	14	15	DQ3

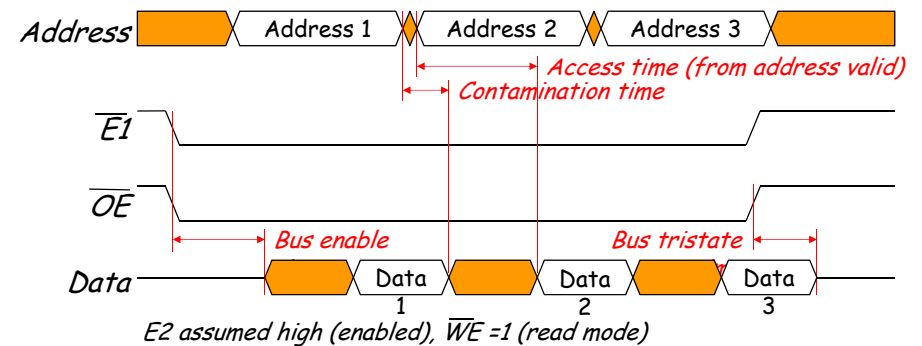


## Reading an Asynchronous SRAM



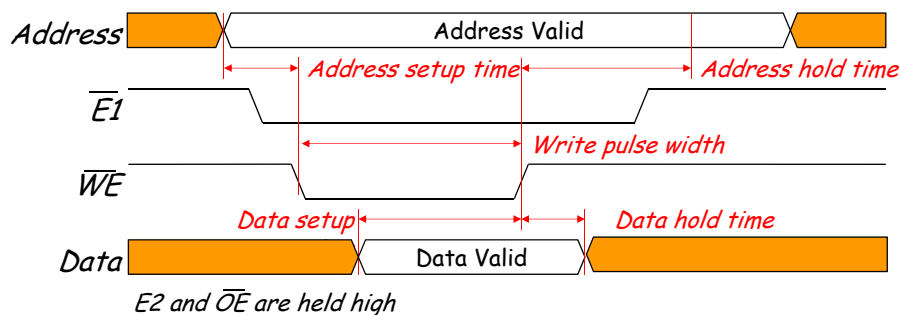
- Read cycle begins when all enable signals ( $\overline{E1}$ ,  $\overline{E2}$ ,  $\overline{OE}$ ) are active
- Data is valid after read access time
  - Access time is indicated by full part number: MCM6264CP-12  $\rightarrow$  12ns
- Data bus is tristated shortly after  $\overline{OE}$  or  $\overline{E1}$  goes high

## Address Controlled Reads



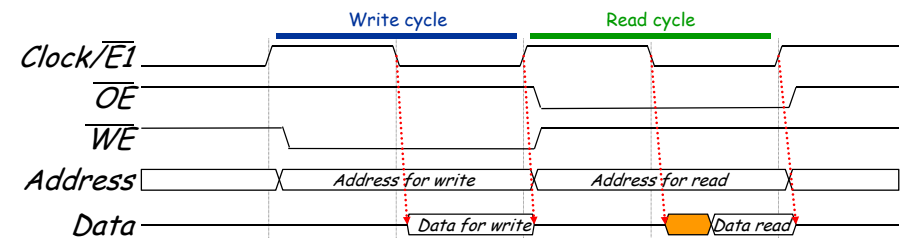
- Can perform multiple reads without disabling chip
- Data bus follows address bus, after some delay

## Writing to Asynchronous SRAM



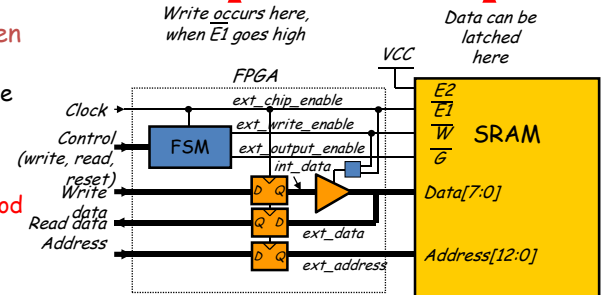
- Data latched when  $\overline{WE}$  or  $\overline{E1}$  goes high (or  $\overline{E2}$  goes low)
  - Data must be stable at this time
  - Address must be stable before  $\overline{WE}$  goes low
- Write waveforms are more important than read waveforms
  - Glitches to address can cause writes to random addresses!

## Sample Memory Interface Logic



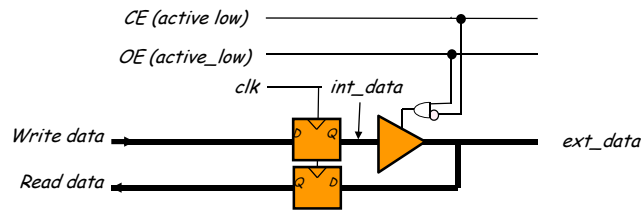
Drive data bus **only** when clock is low

- Ensures address are stable for writes
- Prevents bus contention
- Minimum clock period is twice memory access time





## Tristate Data Buses in Verilog



```

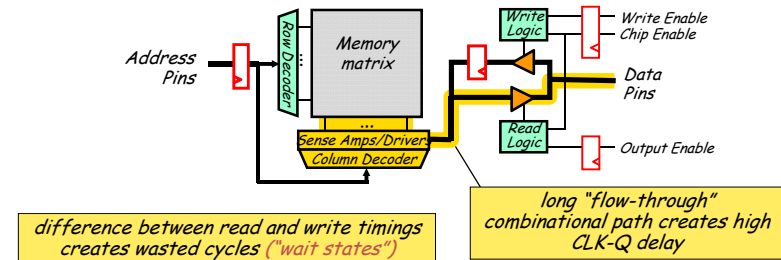
output CE,OE; // these signals are active low
inout [7:0] ext_data;
reg [7:0] read_data,int_data
wire [7:0] write_data;

always @(posedge clk) begin
    int_data <= write_data;
    read_data <= ext_data;
end

// Use a tristate driver to set ext_data to a value
assign ext_data = (~CE & OE) ? int_data : 8'hZZ;
    
```

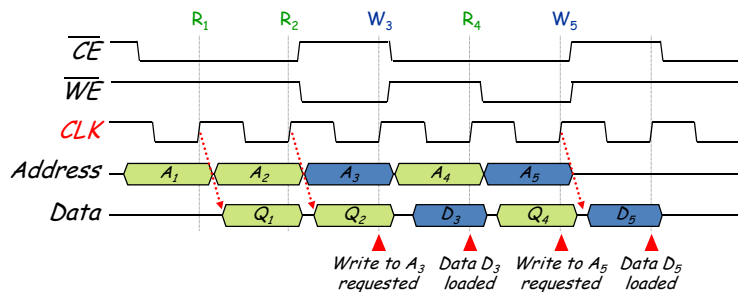
## Synchronous SRAM Memories

- **Clocking** provides input synchronization and encourages more reliable operation at high speeds



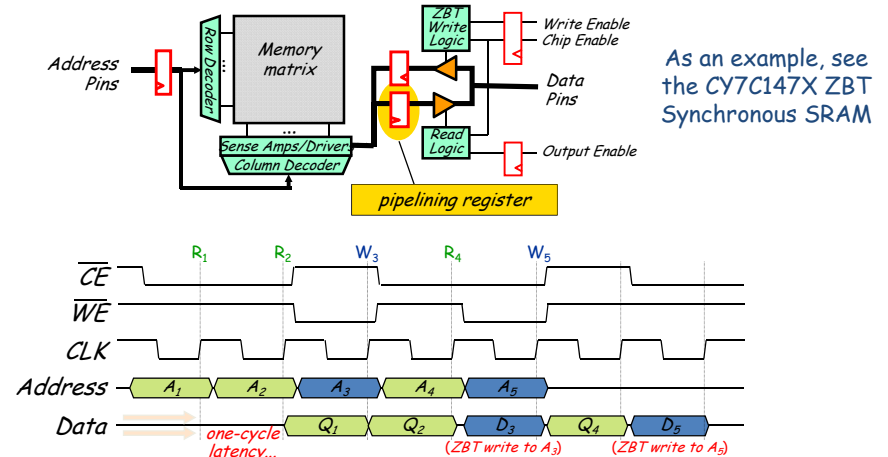
## ZBT Eliminates the Wait State

- The wait state occurs because:
  - On a read, data is available *after* the clock edge
  - On a write, data is set up *before* the clock edge
- ZBT ("zero bus turnaround") memories **change the rules for writes**
  - On a write, data is set up *after* the clock edge (so that it is read on the following edge)
  - Result: no wait states, higher memory throughput



## Pipelining Allows Faster CLK

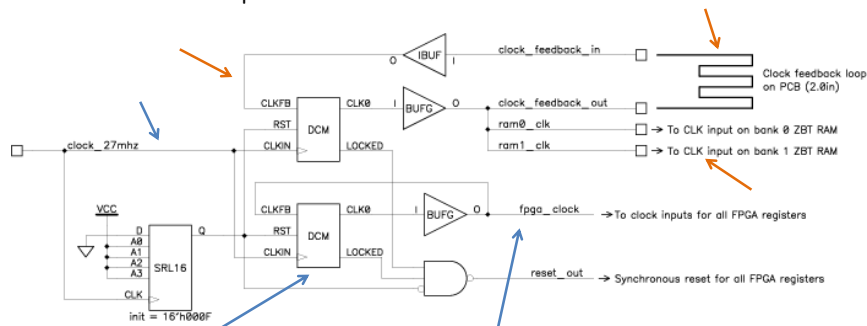
- Pipeline the memory by registering its output
  - Good: Greatly reduces CLK-Q delay, allows higher clock (more throughput)
  - Bad: Introduces an extra cycle before data is available (more latency)



As an example, see the CY7C147X ZBT Synchronous SRAM

## Labkit ZBT interface

The upper DCM is used to generate the de-skewed clock for the external ZBT memories. The feedback loop for this DCM includes a 2.0 inch long trace on the labkit PCB and matches in distance all of the PCB traces from the FPGA to the ZBT memories. The propagation delay from the output of the upper DCM back to its CLKFB input should be almost exactly the same as the propagation delay from the DCM output to the ZBT memories.

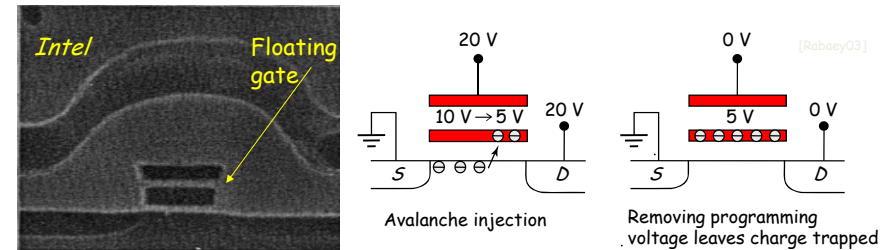


The lower DCM is used to ensure that the fpga\_clock signal, which clocks all of the FPGA flip-flops, is in phase with the reference clock (clock\_27mhz).

## EEPROM

Electrically Erasable Programmable Read-Only Memory

### EEPROM - The Floating Gate Transistor



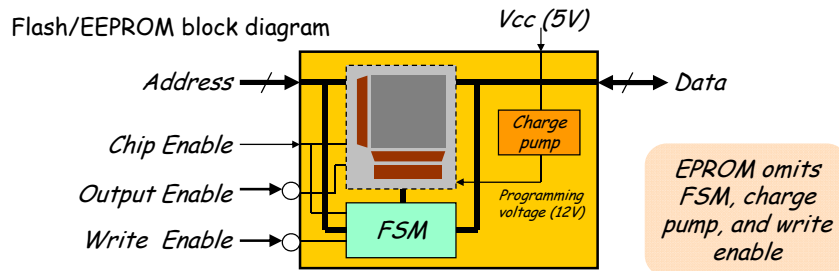
This is a non-volatile memory (retains state when supply turned off)

Usage: Just like SRAM, but **writes are much slower** than reads  
(write sequence is controlled by an FSM internal to chip)

Common application: configuration data (serial EEPROM)

## Interacting with Flash and (E)EPROM

- Reading from flash or (E)EPROM is the same as reading from SRAM
- Vpp: input for programming voltage (12V)
  - EPROM: Vpp is supplied by programming machine
  - Modern flash/EEPROM devices generate 12V using an on-chip charge pump
- EPROM lacks a write enable
  - Not in-system programmable (must use a special programming machine)
- For flash and EEPROM, write sequence is controlled by an internal FSM
  - Writes to device are used to send signals to the FSM
  - Although the same signals are used, one can't write to flash/EEPROM in the same manner as SRAM

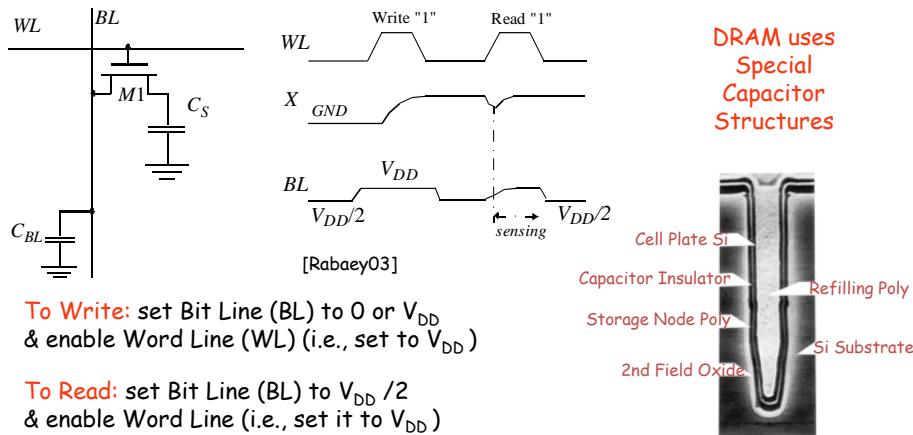


## Flash Memory - Nitty Gritty

- Flash memory uses NOR or NAND flash.
  - NAND cells connected in series like resembling NAND gate.
  - NAND requires 60% of the area compared to NOR. NAND used in flash drives.
  - Endurance: 100,000 - 300,000 p/e cycles
  - Life cycle extended through wear -leveling: mapping of physical blocks changes over time.
- Flash memory limitations
  - Can be read or written byte at a time
  - Can only be erased block at a time
  - Erase sets bits to 1.
  - Location can be re-written if the new bit is zero.
- Labkit has 128Mbits of memory in 1Mbit blocks.
  - 3 Volt Intel StrataFlash® Memory (28F128J3A)
  - 100,000 min erase cycle per block
  - Block erasures takes one second
  - 15 minutes to write entire flash ROM

[http://www.embeddedintel.com/special\\_features.php?article=124](http://www.embeddedintel.com/special_features.php?article=124)

## Dynamic RAM (DRAM) Cell

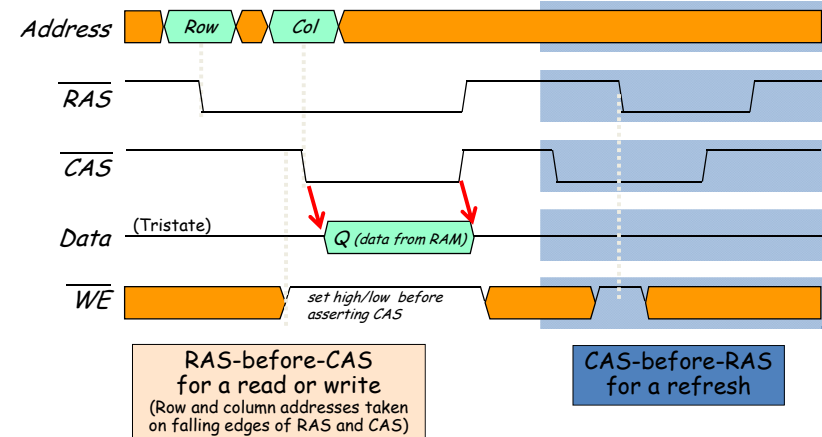


**To Write:** set Bit Line (BL) to 0 or  $V_{DD}$  & enable Word Line (WL) (i.e., set to  $V_{DD}$ )

**To Read:** set Bit Line (BL) to  $V_{DD}/2$  & enable Word Line (i.e., set it to  $V_{DD}$ )

- DRAM relies on charge stored in a capacitor to hold state
- Found in all high density memories (one bit/transistor)
- Must be "refreshed" or state will be lost - high overhead

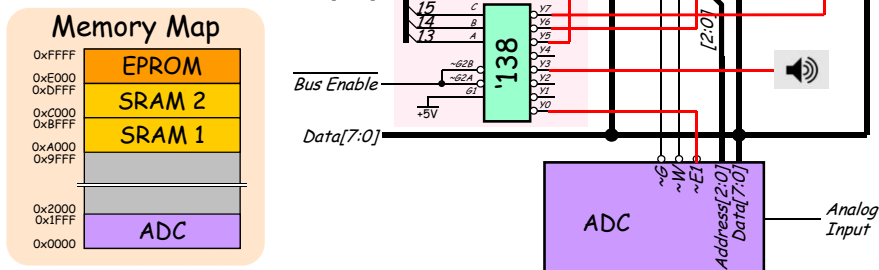
## Asynchronous DRAM Operation



- Clever manipulation of RAS and CAS after reads/writes provide more efficient modes: early-write, read-write, hidden-refresh, etc. (See datasheets for details)

## Addressing with Memory Maps

- Address decoder selects memory
  - Example: '138 3-to-8 decoder
  - Produces enable signals
- SRAM-like interface often used for peripherals
  - Known as "memory mapped" peripherals



## Memory Devices: Helpful Knowledge

- SRAM vs. DRAM**
  - SRAM holds state as long as power supply is turned on. DRAM must be "refreshed" - results in more complicated control
  - DRAM has much higher density, but requires special capacitor technology.
  - FPGA usually implemented in a standard digital process technology and uses SRAM technology
- Non-Volatile Memory**
  - Fast Read, but very slow write (EPROM must be removed from the system for programming!)
  - Holds state even if the power supply is turned off
  - Flash memory is slow, microsecond read, much longer writes
- Memory Internals**
  - Has quite a bit of analog circuits internally -- pay particular attention to noise and PCB board integration
- Device details**
  - Don't worry about them, wait until 6.012 or 6.374

## Memory

- control signals such as *Write Enable* should be registered
- a multi-cycle read/write is safer from a timing perspective than the single cycle read/write approach
- it is a bad idea to enable two tri-states driving the bus at the same time
- an SRAM does not need to be "refreshed" while a DRAM requires refresh
- an EPROM/EEPROM/FLASH cell can hold its state even if the power supply is turned off
- a synchronous memory can result in higher throughput

## Labkit Memory

- Regular registers in logic blocks
  - Operates at system clock speed, expensive (CLB utilization)
  - Configuration set by Verilog design (eg FIFO, single/dual port, etc)
- FPGA Distributed memory
  - Operates at system clock speed
  - Uses LUTs (16 bits) for implementation, expensive (CLB utilization)
  - Requires significant routing for implementation
  - Configured using CoreGen
  - Theoretical maximum: 1Mbit
- FPGA block ram:
  - Implemented with (18 kbit) dedicated memory blocks distributed throughout the FPGA
  - Pipelined (clocked) operations
  - Labkit XCV2V6000: 144 BRAMs, 2952K bits total
- ZBT SRAM
  - two synchronous, 512k x 36 ZBT SRAM chips
  - Operates up to 167MHz
- Flash memory
  - 128Mbits with 100,000 minimum erase cycle per block
  - Slow read access, even slower write access time!
  - Must cache to ZBT or BRAM for video display

## Nexys4 DDR Memory

- Regular registers in logic blocks
  - Operates at system clock speed, expensive (CLB utilization)
  - Configuration set by Verilog design (eg FIFO, single/dual port, etc)
- FPGA Distributed memory
  - Operates at system clock speed
  - Uses LUTs (16 bits) for implementation, expensive (CLB utilization)
  - Requires significant routing for implementation
  - Configured using IP
  - Theoretical maximum: 1Mbit
- FPGA block ram:
  - 4,860K bits total
- DDR2 SDRAM
  - 128MiB (Megabytes)
  - Requires MIG (Memory Interface Generator) Wizard
- Flash memory
  - 16MiB
  - Slow read access, even slower write access time!
- microSD port
  - Tested with 2GB (Windows 7, FPGA)