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
  – Almost cost competitive with hard disks: 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 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 #(.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: 16x(1,2,4,8), 32x(1,2,4,8), 64x(1,2), 128x1
  – Dual port (1 R/W, 1R): 16x1, 32x1, 64x1
  – 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: 16Kx1, 8Kx2, 4Kx4
    with parity: 2Kx(8+1), 1Kx(16+2), 512x(32+4)
  – Single or dual port
  – Pipelined (clocked) operations
  – Labkit XCV2V6000: 144 BRAMs, 2952K bits total
LUT-based RAMs

Slice Distributed RAM Diagram

CLB Distributed RAM Switching Characteristics

<table>
<thead>
<tr>
<th>Description</th>
<th>Symbol</th>
<th>Speed Grade</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential Delays</strong></td>
<td></td>
<td>-6</td>
<td>-5</td>
</tr>
<tr>
<td>Clock CLK to X/Y outputs (WE active) in 16 x 1 mode</td>
<td>(T_{SHOCK16})</td>
<td>1.63</td>
<td>1.79</td>
</tr>
<tr>
<td>Clock CLK to X/Y outputs (WE active) in 32 x 1 mode</td>
<td>(T_{SHOCK32})</td>
<td>1.97</td>
<td>2.17</td>
</tr>
<tr>
<td>Clock CLK to F5 output</td>
<td>(T_{SHOCKF5})</td>
<td>1.77</td>
<td>1.94</td>
</tr>
<tr>
<td><strong>Setup and Hold Times Before/After Clock CLK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BX/BY data inputs (DIN)</td>
<td>(T_{\text{DS/DM}})</td>
<td>0.53/0.09</td>
<td>0.58/0.10</td>
</tr>
<tr>
<td>F/G address inputs</td>
<td>(T_{\text{AS/TAN}})</td>
<td>0.40/0.00</td>
<td>0.44/0.00</td>
</tr>
<tr>
<td>SR input (WS)</td>
<td>(T_{\text{WES/TWEH}})</td>
<td>0.42/0.01</td>
<td>0.46/0.01</td>
</tr>
<tr>
<td><strong>Clock CLK</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minimum Pulse Width, High</td>
<td>(T_{WPH})</td>
<td>0.57</td>
<td>0.63</td>
</tr>
<tr>
<td>Minimum Pulse Width, Low</td>
<td>(T_{WPL})</td>
<td>0.57</td>
<td>0.63</td>
</tr>
<tr>
<td>Minimum clock period to meet address write cycle time</td>
<td>(T_{WC})</td>
<td>1.14</td>
<td>1.25</td>
</tr>
<tr>
<td><strong>Combinatorial Delays</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-input function: F/G inputs to X/Y outputs</td>
<td>(T_{\text{IL0}})</td>
<td>0.35</td>
<td>0.39</td>
</tr>
</tbody>
</table>
LUT-based RAM Modules

// 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]));
Tools will often build these for you...

From Lab 2:

```verilog
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'b1011011;
    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'b1011100;
    4'hE: segments[6:0] = 7'b1111100;
    4'hF: segments[6:0] = 7'b1110001;
    default: segments[6:0] = 7'b00000000;
  endcase
  segments[7] = 1'b0; // decimal point
end
```

---

```
=============================================   HDL Synthesis   ---------------------------------------------
*            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>
 Gate Net
-----------------------------------------------
IBUF:in->out fanout Delay Delay Logical Name
-----------------------------------------------
switch_3_IBUF: I->O 7 0.825 1.102
Mrom__n0000_inst_lut4_01: I->O 1 0.439 0.517
user1_0_OBUF: I->O 4.361
--------------------------------------------------------------------------------
Total  7.244ns (5.625ns logic, 1.619ns route)
(77.7% logic, 22.3% route)
```

---

6.111 Fall 2017  Lecture 12
Block Memories (BRAMs)

\[ (W_{\text{DATA}} + W_{\text{PARITY}}) \times \text{(LOCATIONS)} = 18K \text{ bits} \]

<table>
<thead>
<tr>
<th>Dual-Port Block RAM Primitives</th>
<th>Port A Width</th>
<th>Port B Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMB16_S1_S1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S1_S2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S1_S4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S1_S9</td>
<td>(8+1)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S1_S18</td>
<td>(16+2)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S1_S36</td>
<td>(32+4)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S2_S2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S2_S4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S2_S9</td>
<td>(8+1)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S2_S18</td>
<td>(16+2)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S2_S36</td>
<td>(32+4)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S4_S4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S4_S9</td>
<td>(8+1)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S4_S18</td>
<td>(16+2)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S4_S36</td>
<td>(32+4)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S9_S9</td>
<td>(8+1)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S9_S18</td>
<td>(16+2)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S9_S36</td>
<td>(32+4)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S18_S18</td>
<td>(16+2)</td>
<td></td>
</tr>
<tr>
<td>RAMB16_S18_S36</td>
<td>(32+4)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Single-Port Block RAM Primitives</th>
<th>Port Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAMB16_S1</td>
<td>1</td>
</tr>
<tr>
<td>RAMB16_S2</td>
<td>2</td>
</tr>
<tr>
<td>RAMB16_S4</td>
<td>4</td>
</tr>
<tr>
<td>RAMB16_S9</td>
<td>(8+1)</td>
</tr>
<tr>
<td>RAMB16_S18</td>
<td>(16+2)</td>
</tr>
<tr>
<td>RAMB16_S36</td>
<td>(32+4)</td>
</tr>
</tbody>
</table>

6.111 Fall 2017  Lecture 12
BRAM Operation

Source: Xilinx App Note 463
BRAM timing

Block SelectRAM Timing Diagram

Block SelectRAM Switching Characteristics

<table>
<thead>
<tr>
<th>Description</th>
<th>Symbol</th>
<th>Speed Grade</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential Delays</td>
<td></td>
<td>-6</td>
<td>-5</td>
</tr>
<tr>
<td>Clock CLK to DOUT output</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Setup and Hold Times Before Clock CLK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR inputs</td>
<td></td>
<td>0.29/ 0.00</td>
<td>0.32/ 0.00</td>
</tr>
<tr>
<td>DIN inputs</td>
<td></td>
<td>0.20/ 0.00</td>
<td>0.32/ 0.00</td>
</tr>
<tr>
<td>EN input</td>
<td></td>
<td>0.95/ 0.46</td>
<td>1.04/ 0.50</td>
</tr>
<tr>
<td>RST input</td>
<td></td>
<td>1.31/ 0.71</td>
<td>1.44/ 0.78</td>
</tr>
<tr>
<td>WEN input</td>
<td></td>
<td>0.57/ 0.19</td>
<td>0.63/ 0.21</td>
</tr>
<tr>
<td>Clock CLK</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLKA to CLKB setup time for different ports</td>
<td></td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Minimum Pulse Width, High</td>
<td></td>
<td>1.17</td>
<td>1.29</td>
</tr>
<tr>
<td>Minimum Pulse Width, Low</td>
<td></td>
<td>1.17</td>
<td>1.29</td>
</tr>
</tbody>
</table>
Using BRAMs (eg, a 64Kx8 ram)

- From menus: Project → New Source…

Select “IP”

Fill in name

Click “Next” when done…
BRAM Example

Click open folders

Select "Single Port Block Memory"

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

Fill in name (again?!)

Select RAM vs ROM

Fill in width & depth

Usually “Read After Write” is what you want

Click “Next” …
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:
  
  ```verilog
  ram64x8 foo(.addr(addr), .clk(clk), .we(we), .din(din), .dout(dout));
  ```
Memory Classification & Metrics

Key Design Metrics:
1. Memory Density (number of bits/mm²) and Size
2. Access Time (time to read or write) and Throughput
3. Power Dissipation
Static RAMs: Latch Based 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

$2^{L-K}\times M$ memory

Small cells $\rightarrow$ small mosfets $\rightarrow$ small dV on bit line

Row Decode

$A_0$ $A_{K-1}$ $A_K$ $A_{K+1}$ $\cdots$ $A_{L-1}$

2$^L$-K

Bit Line

Storage Cell

Word Line

$2^{L-K}$ row by $M\times2^K$ column cell array

Sense Amps/Driver

Amplify swing to rail-to-rail amplitude

Column Decode

Selects appropriate word (i.e., multiplexer)

Input-Output (M bits)
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

Write: Set BL, BL to (0, V_{DD}) or (V_{DD}, 0) then enable WL (= V_{DD})

Read: Disconnect drivers from BL and BL, then enable WL (= V_{DD}). Sense a small change in BL or BL
Using External Memory Devices

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

Concept of “Data Bus”

- 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”
MCM6264C 8K x 8 Static RAM

On the outside:

Address
Chip Enables \( \overline{E1} \) \( \overline{E2} \)
Write Enable \( \overline{WE} \)
Output Enable \( \overline{OE} \)

Data \( \text{DQ}[7:0] \)

MCM6264C

Chip Enables (\( E1 \) and \( E2 \))
\( E1 \) must be low and \( E2 \) must be high to enable the chip

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

Output Enable (\( OE \) or \( G \))
When low (and chip is enabled), data bus is driven with value of selected memory location

On the inside:

Memory matrix
256 rows
32 Column

A2 \( \rightarrow \) A3 \( \rightarrow \) A4 \( \rightarrow \) A5 \( \rightarrow \) A7 \( \rightarrow \) A8 \( \rightarrow \) A9 \( \rightarrow \) A11

Row Decoder

Sense Amps/Drivers

Column Decoder

DQ[7:0]

\( \overline{E1} \) \( \overline{E2} \)

\( \overline{WE} \)

\( \overline{OE} \)

Pinout

NC 1
A12 2
A7 3
A6 4
A5 5
A4 6
A3 7
A2 8
A1 9
A0 10
D11 11
D12 12
D13 13
VSS 14
VCC 15
W 26
E1 27
E2 28
A8 29
A9 30
A11 31
G 32
ET 33
DQ7 34
DQ6 35
DQ5 36
DQ4 37
DQ3 38

6.111 Fall 2017
Lecture 12
Reading an Asynchronous SRAM

- Read cycle begins when all enable signals (E1, E2, OE) are active.

- Data is valid after read access time:
  - Access time is indicated by full part number: $MCM6264CP-12 \rightarrow 12\text{ns}$

- Data bus is tristated shortly after OE or 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 \( \overline{EI} \) goes high (or 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

Write occurs here, when $E_1$ goes high

Data can be latched here
Tristate Data Buses in Verilog

```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

- Pipe 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/EEPROM block diagram
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 a time
  – Can only be erased block at a time
  – Erasure 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
Dynamic RAM (DRAM) Cell

- 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

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 \( V_{DD} \))

[Image showing the DRAM cell structure and waveforms for write and read operations.

[Rabaey03]
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 (e.g., 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)
• Upload project files to course website: one per team
• Lab 5 due Mon 9P
• Meet with staff for project ideas