Digital Design: An Embedded Systems Approach Using Verilog

Chapter 5
Memories

Portions of this work are from the book, Digital Design: An Embedded Systems Approach Using Verilog, by Peter J. Ashenden, published by Morgan Kaufmann Publishers, Copyright 2007 Elsevier Inc. All rights reserved.
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

\[ \begin{array}{c}
0 \\
1 \\
2 \\
3 \\
4 \\
5 \\
6 \\
\end{array} \quad \begin{array}{c}
2^{n-2} \\
2^{n-1} \\
\end{array} \]
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 = 1,024K = 1Mbit
  - Requires 15 address bits

- Size is determined by application requirements
### Basic Memory Operations

- **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:** \( en = 0 \)
Wider Memories

- Memory components have a fixed width
  - E.g., $\times 1$, $\times 4$, $\times 8$, $\times 16$, ...

- Use memory components in parallel to make a wider memory
  - E.g, three $16K\times 16$ components for a $16K\times 48$ 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
• Example: 64K×8 memory composed of 16K×8 components
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
### Switching Characteristics Over the Operating Range

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>-12 Min.</th>
<th>-12 Max.</th>
<th>-15 Min.</th>
<th>-15 Max.</th>
<th>-17 Min.</th>
<th>-17 Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\textsubscript{RC}</td>
<td>Read Cycle Time</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{AA}</td>
<td>Address to Data Valid</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{OHA}</td>
<td>Data Hold from Address Change</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{ACE}</td>
<td>CE LOW to Data Valid</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{DOE}</td>
<td>OE LOW to Data Valid</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

### WRITE CYCLE

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>-12 Min.</th>
<th>-12 Max.</th>
<th>-15 Min.</th>
<th>-15 Max.</th>
<th>-17 Min.</th>
<th>-17 Max.</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>t\textsubscript{WC}</td>
<td>Write Cycle Time</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{SCE}</td>
<td>CE LOW to Write End</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{AW}</td>
<td>Address Set-Up to Write End</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{HA}</td>
<td>Address Hold from Write End</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{SA}</td>
<td>Address Set-Up to Write Start</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{PWE}</td>
<td>WE Pulse Width</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{SD}</td>
<td>Data Set-Up to Write End</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{HD}</td>
<td>Data Hold from Write End</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{LZWE}</td>
<td>WE HIGH to Low Z\textsuperscript{[6]}</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{H2WE}</td>
<td>WE LOW to High Z\textsuperscript{[5, 6]}</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>t\textsubscript{BW}</td>
<td>Byte Enable to End of Write</td>
<td>10</td>
<td>12</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>
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
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

![Timing diagram]

```verilog
M(a_2)
```
Memories in Verilog

- RAM storage represented by an array variable

```verilog
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

```verilog
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

```verilog
always @(posedge clk) // State register
...
always @* // Next-state logic
...
always @* begin // Output logic
...
endmodule
```
Pipelined SSRAM in Verilog

```verilog
going 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
```
Example: RAM Core Generator

![Block Memory Generator v1.1](image)

Digital Design — Chapter 5 — Memories