MEMORIES
Memory in Digital Systems

• Three primary components of digital systems
  – Datapath (does the work)
  – Control (manager)
  – Memory (storage)
Memories

- Use in general digital processors
  - Instructions
  - Data
- Uses more common in digital signal processors
  - Buffering input/intermediate/output data (e.g., rate matching)
  - Storing fixed numbers (e.g., coefficients)
  - Often relatively small (e.g., 8-64-256 words) and numerous (dozens are not unusual)
Memory View 1: Major Types

• Read-write memories
  – SRAM: static memory
    • Data is stored as the state of a bistable circuit
    • State is retained without refresh as long as power is supplied
  – DRAM: dynamic memory
    • Data is stored as a charge on a capacitor
    • State leaks away, refresh is required

• NVRWM: non-volatile read-write memory
  – Flash: ROM at low voltages, writable at high voltages
  – EPROM: ROM, but erasable with UV light

• ROM: read-only memory
  – Non-volatile—mask programmed at manufacture
  – Fuses—links blown during manufacture
  – Anti-fuses—links made during manufacture
Memory View 2: Logical Categories

• Combinational (output depends on present inputs only)
  – ROM: read-only memory
• Feels like Combinational but technically Sequential
  – PROM: programmable read-only memory
  – EPROM: ROM, but erasable with UV light
• Sequential (output depends on present and past inputs)
  – SRAM: static memory
  – DRAM: dynamic memory
  – Flash: ROM at low voltages, writable at high voltages
Memories

- Memories generally contain several components:
  - Array of cells
  - Word decoder
  - Write circuitry
  - Read circuitry (sense amplifiers)
  - wordlines
  - bitlines

- Interface signals
  - Address (one for each port)
  - Data (one for each port)
  - Enable_write
  - Enable_read (likely)
  - Clock (sometimes)
Memories for custom processors can be built in a number of ways:

1) On-chip “macro” memory arrays
   - Think of as a single giant standard cell
   - FPGAs include them (“block RAMs” or “block memory”)

2) On-chip memory synthesized from standard cells

3) Off-chip memories (perhaps for > approx. 10 MB)
   - Very large DRAM
   - Non-volatile memory such as flash memory
   - (We could also include disks, NAS, cloud, etc.)
1) Memory Macro-cells

- Memory macro-cell generators are available for larger memories
- Typically a software tool generates a large variety of possible memories where a user may select options such as:
  - Number of words
  - Word-width (in bits)
  - Number of read ports
  - Number of write ports
  - Rd/wr or ROM
  - Built-in test circuits
  - Registered inputs and/or outputs
- Tool produces models for verilog, place & route, and other CAD views
1) On-chip “macro” memory arrays

- Generally very area efficient due to dense memory cells (single-ported memories likely use 6-transistor (6T) memory cells)
- Generally good energy efficiency due to low-activity memory array architecture

[T. Nanya, et al., TITAC-2 processor]
2) Synthesized Memory

- Can synthesize memory from standard cells
  - Memory cells are now flip-flops
  - \(clk\) likely routed to all cells
  - Probably best for small memories only
  - Read bitline logic may be muxes
2) Synthesized Memory

- Standard cell layout is typically irregular
  - Wires not shown
  - Clocks routed to each “reg” (flip-flop)
Verilog Memories

• Declaring a 16-bit, 128-word memory
  - reg [15:0] Mem [0:127];

• Reading
  - source1 = Mem[addr_rd];

• Writing
  - Blocking or non-blocking depends on how you want to handle simultaneous reads and writes
  - Mem[addr_wr] <= #1 c_datapath_out; // makes sense for FFs
  - Mem[addr_wr] = c_datapath_out;
Verilog Memories

- This memory performs writes on the positive edge of the clock when `write_enable` is high.
- The output is not controlled by the clock and outputs the correct memory word for any address on `addr_rd`:
  - Picture a large mux tree connecting every word in the memory to the output port.
  - Sometimes called an “asynchronous read”

```verilog
reg [15:0]   mem [0:127];
always @(posedge clk) begin
  if (write_enable == 1'b1) begin
    mem[addr_wr] <= #1 data_in;
  end
end
assign data_out = mem[addr_rd];
```
Verilog Memories

- This memory also performs writes on the positive edge of the clock when write_enable is high.
- But this design contains a “synchronous read” which updates the output port only on the active edge of the clock:
  - There is now one clock cycle of delay from when addr_rd is valid to when the read output is valid.

```verilog
reg [15:0] mem [0:127];
always @(posedge clk) begin
  if (write_enable == 1'b1) begin
    mem[addr_wr] <= #1 data_in;
  end
  data_out <= #1 mem[addr_rd];
end
```
Verilog Memories

• Reading and writing
  – Cannot access a portion of a word without first reading the whole word in many simulators and CAD tools (though it is supported in some tools). Good practice to not do it! So don’t in this class.
  - source1 = Mem[addr_rd][5];  // won’t work sometimes
  - temp   = Mem[addr_rd];      // use these 2 lines instead
    source1 = temp[5];

• Multiple unclocked read ports
  – Simply make individual read statements for each read port
  - data1  = Mem[addr_rd1];
    data2  = Mem[addr_rd2];
    data3  = Mem[addr_rd3];
Memory Pipelining/Timing
Timing Style 1

- **Style 1:** Registers are outside the memory array
- **Memory macros** typically register input addresses and data “outside” the memory array
• For cases when the memory is a combinational block
  – Add pipeline registers outside block as appropriate or as needed to meet the target clock frequency
• For cases when the memory has a built-in register for its outputs
  – Add a pipeline register to the inputs as appropriate or as needed to meet the target clock frequency
• For cases when the memory has a built-in register for its inputs
  – Add a pipeline register to the outputs as appropriate or as needed to meet the target clock frequency
Memory Pipelining/Timing
Timing Style 2

- **Style 2**: Register is *in the middle* of the memory array
- **Standard cell memories** store bits in the array in **FFs**
  - This forces a pipe stage across the middle of the memory array
• Built-in pipeline stage in memory array
• May place logic in memory pipe stages to balance pipeline