ECS 409: Computer Organization – Assignment 3

Adheesh Trivedi

2025-09-13

Problem 1: SR Latch with Enable & Reset

Simple SR latch with async enable and reset; holds state when En=0, reset dominates.

```
`timescale 1ns / 1ps

module SR_LATCH (
  input S, R, En, Rst,
  output reg Q
);
  always @(*) begin
    // Asynchronous Reset dominates
    if (Rst) Q = 0;
    else if (En) begin
      if (S & ~R) begin
        Q = 1;
      end else if (~S & R) begin
        Q = 0;
      end
    end
  end
endmodule


module TEST();
  reg S, R, En, Rst;
  wire Q;

  SR_LATCH uut (S, R, En, Rst, Q);

  initial begin
    $dumpfile("q1.vcd");
    $dumpvars(0, TEST);
    $display("Solution by Adheesh Trivedi");
    $display("============================");

    $display("");
    $display("S R En Rst Q");
    $display("-------------");
    $monitor("%b %b %b  %b   %b", S, R, En, Rst, Q);

    // Asynchronous Reset
    S = 0; R = 0; En = 0; Rst = 1; #5;
    Rst = 0; #5;
    // Set
    S = 1; R = 0; En = 1; #5;
    S = 0; #5;
    // Reset
    S = 0; R = 1; En = 1; #5;
    R = 0; En = 0; #5;
    // Not Enabled
    S = 1; R = 0; #5;
    // Enabled again
    En = 1; #5;
    S = 0; #10;
```

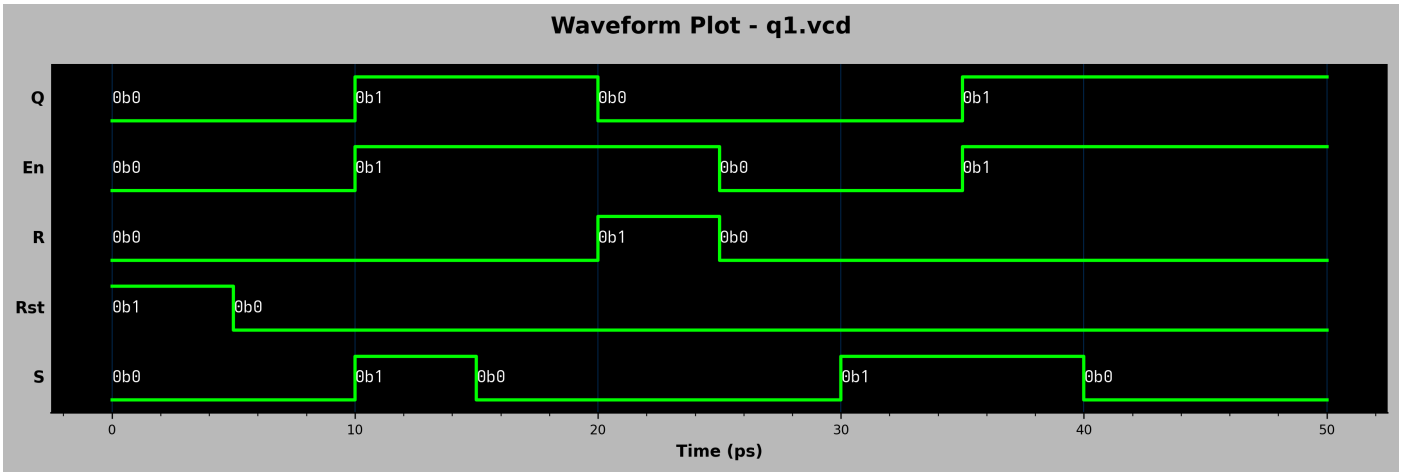
```
  end
endmodule
```

SR latch with En & Rst

1.1 Simulation Output

Q resets when Rst=1 regardless of inputs; when enabled, S=1 sets and R=1 resets. Transitions match table in outputs (stable hold when En=0; no illegal set when R=1).

```
→ vvp q1.vvp
VCD info: dumpfile q1.vcd opened for output.
Solution by Adheesh Trivedi
============================

S R En Rst Q
-------------
0 0 0  1   0
0 0 0  0   0
1 0 1  0   1
0 0 1  0   1
0 1 1  0   0
0 0 0  0   0
1 0 0  0   0
1 0 1  0   1
0 0 1  0   1
```



Problem 2: Improved D Latch with Gate Delays

Behavioral D latch with 1 ns gate delays; shows correct level-sensitive behavior.

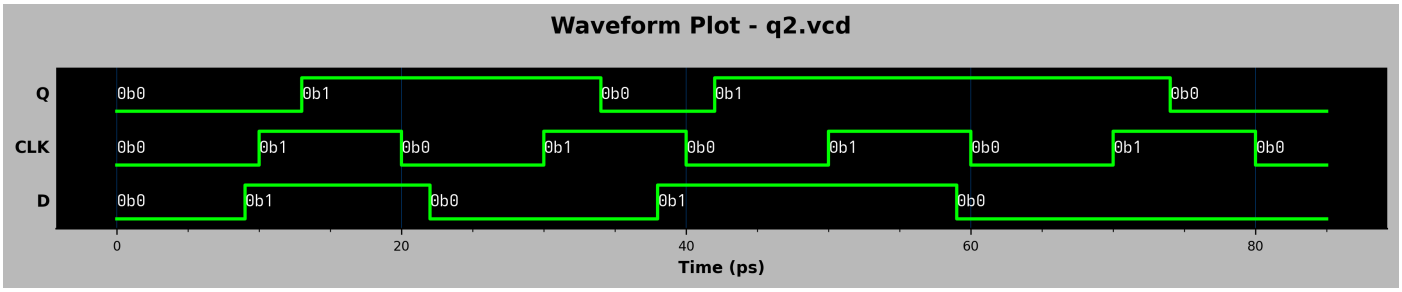
```
`timescale 1ns / 1ps

module d_latch (
  input D, CLK,
  output reg Q
);
  reg N1, N2, not_CLK;

  initial Q = 0;

  always @(D or CLK or not_CLK or N1 or N2 or Q) begin
    #1 N1 = D & CLK;
    #1 not_CLK = ~CLK;
    #1 N2 = not_CLK & Q;
    #1 Q = N1 | N2;
  end

endmodule


module TEST();
  reg D, CLK;
  wire Q;

  d_latch dl (D, CLK, Q);

  initial CLK = 0;
  always begin
    #10; CLK = ~CLK;
  end

  initial begin
    $dumpfile("q2.vcd");
    $dumpvars(0, TEST);
    $display("Solution by Adheesh Trivedi");
    $display("============================");

    $display("");
    $display("Time  D CLK Q");
    $display("---------------");
    $monitor("%5t %b  %b  %b", $time, D, CLK, Q);

    D = 0; #9;
    D = 1; #13;
    D = 0; #16;
    D = 1; #21;
    D = 0; #26;

    $finish;
  end
endmodule
```

Improved D latch (1ns delays)

2.1 Simulation Output

Q follows D only while CLK=1 after 1 ns delay; holds value when CLK=0. Edges and printed times (e.g., 10k, 30k ps) match expected delayed response.

```
→ vvp q2.vvp
VCD info: dumpfile q2.vcd opened for output.
Solution by Adheesh Trivedi
===========================

Time  D CLK Q
--------------
    0 0  0  0
 9000 1  0  0
10000 1  1  0
13000 1  1  1
20000 1  0  1
22000 0  0  1
30000 0  1  1
34000 0  1  0
38000 1  1  0
40000 1  0  0
42000 1  0  1
50000 1  1  1
59000 0  1  1
60000 0  0  1
70000 0  1  1
74000 0  1  0
80000 0  0  0
q2.v:49: $finish called at 85000 (1ps)
```



Problem 3: 3-bit Down Counter (7 to 0)

Counts 7⟶0 and repeats; synchronous reset initializes to 000.

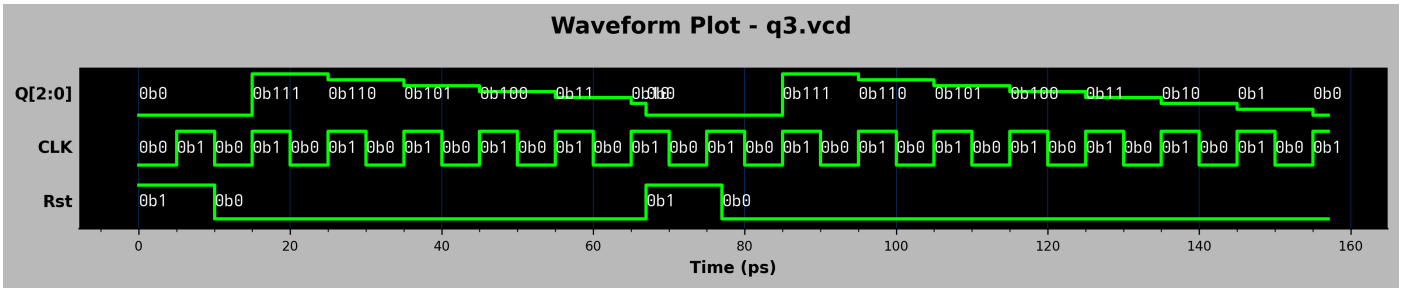
```
`timescale 1ns / 1ps

module COUNTER(
  input wire CLK, Rst,
  output reg [2:0] Q
);
  initial Q = 3'b000;

  always @(posedge CLK or posedge Rst) begin
    // Prioritize reset
    if (Rst) begin
      Q <= 3'b000;
    end else begin
      Q <= Q - 1;
    end
  end
endmodule


module TEST();
  reg CLK, Rst;
  wire [2:0] Q;

  COUNTER uut (CLK, Rst, Q);

  always begin
    #5;
    CLK = ~CLK;
  end

  initial begin
    $dumpfile("q3.vcd");
    $dumpvars(0, TEST);
    $display("Solution by Adheesh Trivedi");
    $display("============================");

    $display("");
    $display("Rst  Q");
    $display("-------");
    $monitor("%2b  %b", Rst, Q);

    CLK = 0; Rst = 1; #10;
    Rst = 0;
    #57;

    Rst = 1; #10;
    Rst = 0; #80;

    $finish;
  end
endmodule
```

Mod-8 down counter

3.1 Simulation Output

Sequence shows 111⟶110⟶…⟶000 then wrap; Rst forces 000 then counts down again. Printed log confirms two full cycles and final wrap as in outputs.txt.

```
→ vvp q3.vvp
VCD info: dumpfile q3.vcd opened for output.
Solution by Adheesh Trivedi
===========================

Rst  Q
-------
 1  000
 0  000
 0  111
 0  110
 0  101
 0  100
 0  011
 0  010
 1  000
 0  000
 0  111
 0  110
 0  101
 0  100
 0  011
 0  010
 0  001
 0  000
q3.v:49: $finish called at 157000 (1ps)
```



Problem 4: Mod-8 Gray Code Counter with UP

Gray code transitions change one bit per step; UP selects advance or hold.

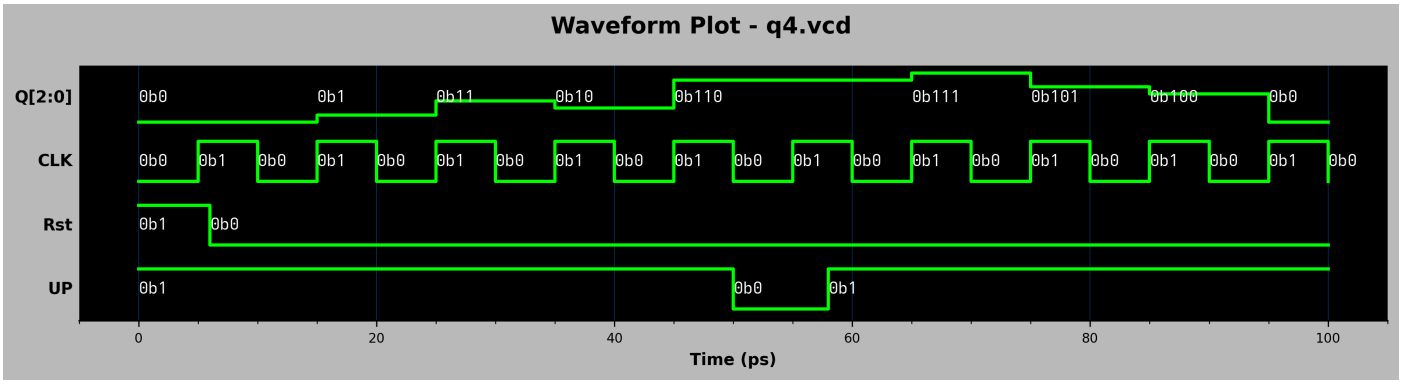
```
`timescale 1ns / 1ps

module GRAY_COUNTER (
  input CLK, input Rst, input UP,
  output reg [2:0] Q
);
  initial Q = 3'b000;

  always @(posedge CLK or posedge Rst) begin
    if (Rst) begin
      Q <= 3'b000;
    // Derived from K-MAPS
    end else if (UP) begin
      Q[0] <= ~ (Q[1] ^ Q[2]);
      Q[1] <= Q[0] ? ~Q[2] : Q[1];
      Q[2] <= Q[0] ? Q[2] : Q[1];
    end
  end
endmodule


module TEST();
  reg CLK, Rst, UP;
  wire [2:0] Q;

  GRAY_COUNTER gc (CLK, Rst, UP, Q);

  initial CLK = 0;

  always begin
    #5; CLK = ~CLK;
  end

  initial begin
    $dumpfile("q4.vcd");
    $dumpvars(0, TEST);
    $display("Solution by Adheesh Trivedi");
    $display("============================");

    $display("");
    $display("UP Q");
    $display("------");
    $monitor("%b  %b", UP, Q);

    Rst = 1; UP = 1; #6;
    Rst = 0; #44;
    UP = 0; #8;
    UP = 1; #42;

    $finish;
  end
endmodule
```

UP/DOWN Gray code counter

4.1 Simulation Output

Logs show single-bit changes and holds when UP=0; wrap-around behavior verified. Waveform matches printed sequence (e.g., 000⟶001⟶011⟶010⟶110…).

```
→ vvp q4.vvp
VCD info: dumpfile q4.vcd opened for output.
Solution by Adheesh Trivedi
===========================

UP Q
------
1  000
1  001
1  011
1  010
1  110
0  110
1  110
1  111
1  101
1  100
1  000
q4.v:50: $finish called at 100000 (1ps)
```



Problem 5: Parameterized N-bit Bidirectional Shift Register

Shift left/right under DIR with enable; parameter N configures width.

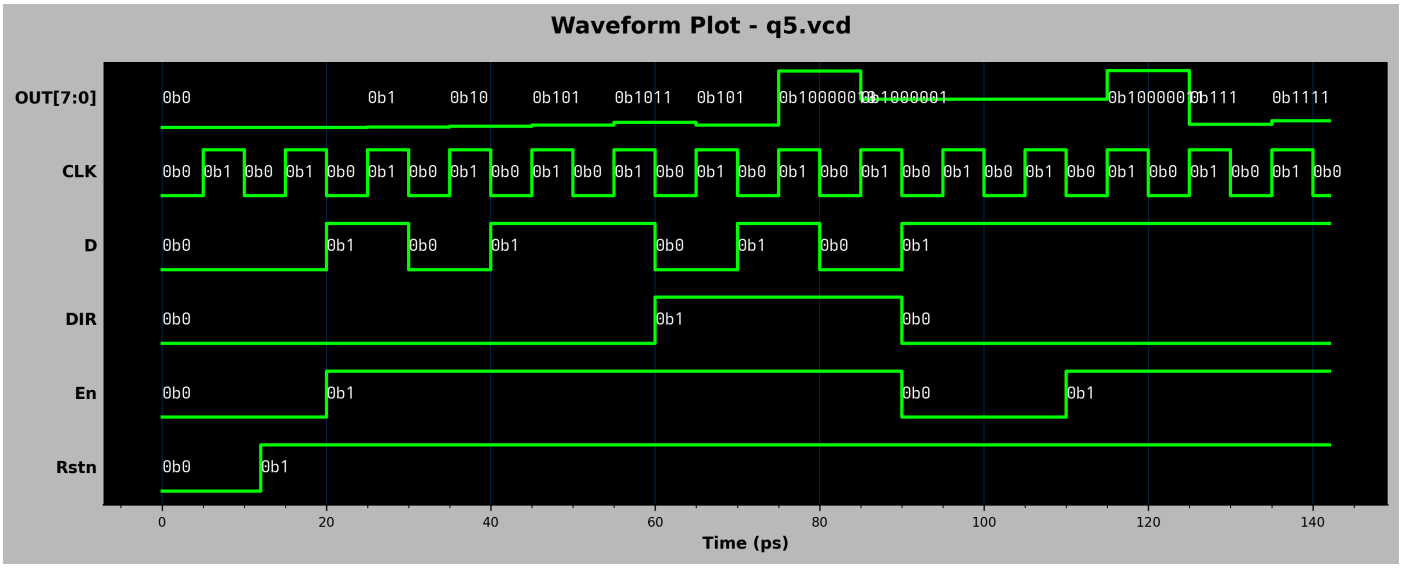
```
`timescale 1ns / 1ps

module N_BIT_BIDIR_SHIFT_REG #(parameter MSB = 8) (
  input wire D, CLK, En, DIR, Rstn,
  output reg [MSB-1:0] OUT
);
  initial OUT = {MSB{1'b0}};

  always @(posedge CLK) begin
    if (!Rstn) begin
     OUT <= {MSB{1'b0}};
    end else if (En) begin
     case (DIR)
       1'b0: OUT <= {OUT[MSB-2:0], D};
        1'b1: OUT <= {D, OUT[MSB-1:1]};
      endcase
    end
  end
endmodule


module TEST();

  reg D, CLK, En, DIR, Rstn;
 wire [7:0] OUT;

 N_BIT_BIDIR_SHIFT_REG #(8) nbbsr (
   D, CLK, En, DIR, Rstn, OUT
 );

  initial CLK = 0;
 always begin
   #5; CLK = ~CLK;
  end

 initial begin
    $dumpfile("q5.vcd");
   $dumpvars(0, TEST);
    $display("Solution by Adheesh Trivedi");
    $display("============================");

   $display("");
    $display("Time   CLK Rstn En DIR D   OUT");
    $display("-----------------------------------");
   $monitor("%6t %b    %b   %b  %b   %b   %b", $time, CLK, Rstn, En, DIR, D, OUT);

   // Reset low for a couple of clock edges (synchronous reset)
   D = 0; En = 0; DIR = 0; Rstn = 0; #12;
   Rstn = 1; #8;

   // Enable and shift left (DIR=0) while feeding data bits
   En = 1; DIR = 0;
   D = 1; #10;
    D = 0; #10;
    D = 1; #10;
    D = 1; #10;
```

```
    // Shift right (DIR=1) with new data pattern
    DIR = 1;
    D = 0; #10;
     D = 1; #10;
     D = 0; #10;

    // Disable shifting; output should hold
     En = 0; D = 1; DIR = 0; #20;

     // Re-enable and shift left again
     En = 1; #32;

     $finish;
  end
endmodule
```

Parametric bidirectional shifter

5.1 Simulation Output

Output shifts as DIR changes and En toggles; patterns match table in log. Reset deassert then enable drives series of shifts, mirroring outputs.txt.



```
→ vvp q5.vvp
VCD info: dumpfile q5.vcd opened for output.
Solution by Adheesh Trivedi
===========================

Time   CLK Rstn En DIR D   OUT
---------------------------------
     0 0    0   0  0   0   00000000
  5000 1    0   0  0   0   00000000
 10000 0    0   0  0   0   00000000
 12000 0    1   0  0   0   00000000
 15000 1    1   0  0   0   00000000
 20000 0    1   1  0   1   00000000
 25000 1    1   1  0   1   00000001
 30000 0    1   1  0   0   00000001
 35000 1    1   1  0   0   00000010
 40000 0    1   1  0   1   00000010
 45000 1    1   1  0   1   00000101
 50000 0    1   1  0   1   00000101
 55000 1    1   1  0   1   00001011
 60000 0    1   1  1   0   00001011
 65000 1    1   1  1   0   00000101
 70000 0    1   1  1   1   00000101
 75000 1    1   1  1   1   10000010
 80000 0    1   1  1   0   10000010
 85000 1    1   1  1   0   01000001
 90000 0    1   0  0   1   01000001
 95000 1    1   0  0   1   01000001
100000 0    1   0  0   1   01000001
105000 1    1   0  0   1   01000001
110000 0    1   1  0   1   01000001
115000 1    1   1  0   1   10000011
120000 0    1   1  0   1   10000011
125000 1    1   1  0   1   00000111
130000 0    1   1  0   1   00000111
135000 1    1   1  0   1   00001111
140000 0    1   1  0   1   00001111
q5.v:70: $finish called at 142000 (1ps)
```

Problem 6: 128×16 Single-Port Memory

Synchronous write on WE; combinational read returns stored data.

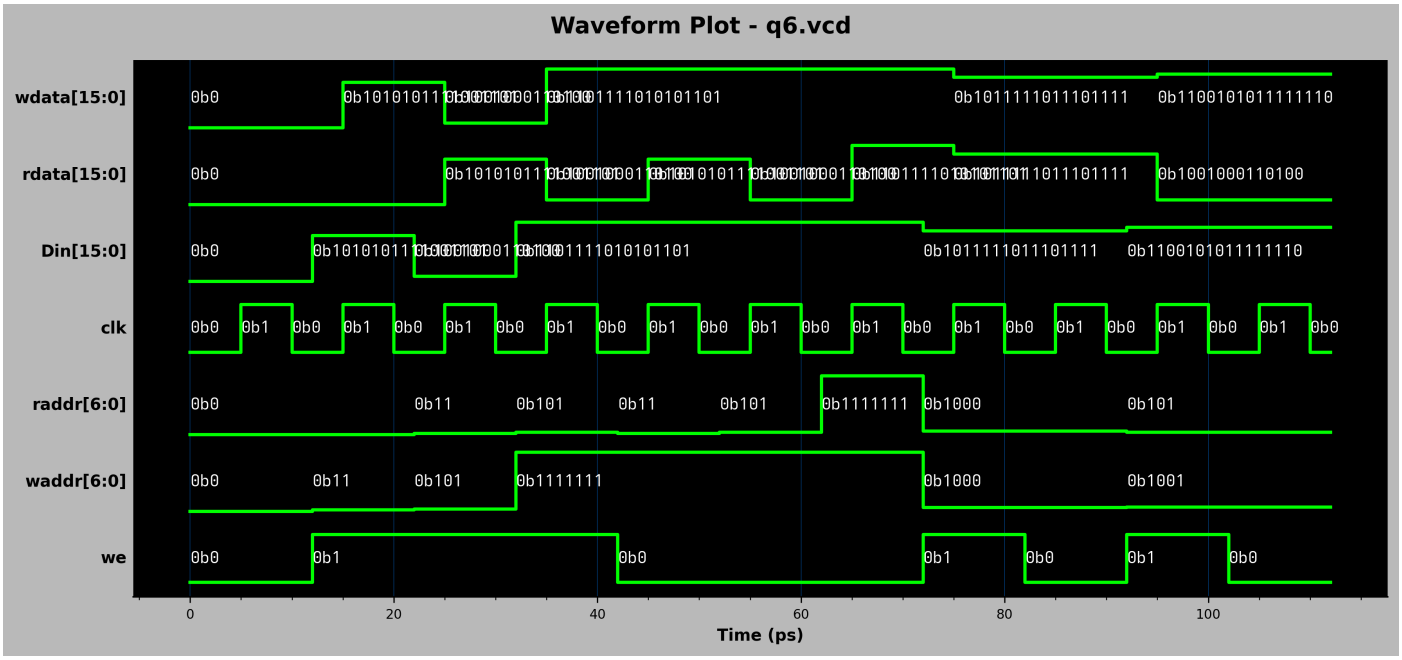
```
`timescale 1ns / 1ps

module MEMORY_1P (
 input  wire clk, we,
  // address
  input  wire [6:0]  waddr, raddr,
  // Data input
 input  wire [15:0] Din,
  // Data output
 output reg  [15:0] rdata, wdata
);
 reg [15:0] mem [0:127];

 integer i;
 initial begin
    // Initialize memory to 0 for determinism
    for (i = 0; i < 128; i = i + 1) begin
      mem[i] = 16'h0000;
   end
    rdata = 16'h0000;
    wdata = 16'h0000;
  end

 always @(posedge clk) begin
    // Write path
    if (we) begin
      mem[waddr] <= Din;
     wdata <= Din;
    end

   // Read path: write-first if raddr == waddr and we is asserted
   if (we && (waddr == raddr)) begin
      rdata <= Din;
    end else begin
     rdata <= mem[raddr];
   end
  end
endmodule


module TEST();
 reg        clk, we;
  reg [6:0]  waddr, raddr;
 reg [15:0] Din;
  wire [15:0] rdata, wdata;

 MEMORY_1P mem (
    clk, we, waddr, raddr, Din, rdata, wdata
 );

  initial clk = 0;
 always begin
   #5; clk = ~clk;
  end

 initial begin
    $dumpfile("q6.vcd");
```

```
    $dumpvars(0, TEST);
     $display("Solution by Adheesh Trivedi");
     $display("============================");

    $display("");
     $display("Time   clk we waddr raddr   Din  | rdata   wdata");
     $display("-------------------------------------------------");
     $monitor("%6t %b   %b  %3d   %3d   0x%04h | 0x%04h  0x%04h", $time, clk, we, waddr, raddr,
Din, rdata, wdata);

     // Initial
    we = 0; waddr = 0; raddr = 0; Din = 16'h0000; #12;

     // Write some locations
     we = 1; waddr = 7'd3;  Din = 16'hABCD; raddr = 7'd0; #10;
     we = 1; waddr = 7'd5;  Din = 16'h1234; raddr = 7'd3; #10;
     we = 1; waddr = 7'd127; Din = 16'hDEAD; raddr = 7'd5; #10;

     // Read back
    we = 0; raddr = 7'd3; #10;
    raddr = 7'd5; #10;
    raddr = 7'd127; #10;

     // Same-cycle write/read to same address -> write-first behavior
    we = 1; waddr = 7'd8; raddr = 7'd8; Din = 16'hBEEF; #10;
    we = 0; raddr = 7'd8; #10;

     // Hold read while writing elsewhere
    raddr = 7'd5; we = 1; waddr = 7'd9; Din = 16'hCAFE; #10;
    we = 0; #10;

     $finish;
  end
endmodule
```

Single-port RAM 128×16

6.1 Simulation Output

Writes at addresses 3,5,127,8,9 then reads back matching data (abcd,1234,dead,beef,cafe). Printed time-stamped table confirms correct read-after-write and later reads.



```
→ vvp q6.vvp
VCD info: dumpfile q6.vcd opened for output.
Solution by Adheesh Trivedi
===========================

Time   clk we waddr raddr   Din   | rdata   wdata
-------------------------------------------------
     0 0   0    0     0   0x0000 | 0x0000  0x0000
  5000 1   0    0     0   0x0000 | 0x0000  0x0000
 10000 0   0    0     0   0x0000 | 0x0000  0x0000
 12000 0   1    3     0   0xabcd | 0x0000  0x0000
 15000 1   1    3     0   0xabcd | 0x0000  0xabcd
 20000 0   1    3     0   0xabcd | 0x0000  0xabcd
 22000 0   1    5     3   0x1234 | 0x0000  0xabcd
 25000 1   1    5     3   0x1234 | 0xabcd  0x1234
 30000 0   1    5     3   0x1234 | 0xabcd  0x1234
 32000 0   1  127     5   0xdead | 0xabcd  0x1234
 35000 1   1  127     5   0xdead | 0x1234  0xdead
 40000 0   1  127     5   0xdead | 0x1234  0xdead
 42000 0   0  127     3   0xdead | 0x1234  0xdead
 45000 1   0  127     3   0xdead | 0xabcd  0xdead
 50000 0   0  127     3   0xdead | 0xabcd  0xdead
 52000 0   0  127     5   0xdead | 0xabcd  0xdead
 55000 1   0  127     5   0xdead | 0x1234  0xdead
 60000 0   0  127     5   0xdead | 0x1234  0xdead
 62000 0   0  127   127   0xdead | 0x1234  0xdead
 65000 1   0  127   127   0xdead | 0xdead  0xdead
 70000 0   0  127   127   0xdead | 0xdead  0xdead
 72000 0   1    8     8   0xbeef | 0xdead  0xdead
 75000 1   1    8     8   0xbeef | 0xbeef  0xbeef
 80000 0   1    8     8   0xbeef | 0xbeef  0xbeef
 82000 0   0    8     8   0xbeef | 0xbeef  0xbeef
 85000 1   0    8     8   0xbeef | 0xbeef  0xbeef
 90000 0   0    8     8   0xbeef | 0xbeef  0xbeef
 92000 0   1    9     5   0xcafe | 0xbeef  0xbeef
 95000 1   1    9     5   0xcafe | 0x1234  0xcafe
100000 0   1    9     5   0xcafe | 0x1234  0xcafe
102000 0   0    9     5   0xcafe | 0x1234  0xcafe
105000 1   0    9     5   0xcafe | 0x1234  0xcafe
110000 0   0    9     5   0xcafe | 0x1234  0xcafe
q6.v:88: $finish called at 112000 (1ps)
```