Lab Report 6

Tuesday 2:15pm

Yen-Jung(Tim) Lu, Antonio Fontan

Introduction

In this lab, we're building and testing a special chip model called SPI SRAM, which stands for Serial Peripheral Interface Static Random-Access Memory. This chip has a two-way communication feature and an extra connection called a secondary SPI interface that makes it easy to link with a microprocessor. SPI, or Serial Peripheral Interface, is like a special highway for data that uses four wires to send and receive information. Our goal is to see how well this chip works, especially in talking to a microprocessor.

Procedure

Describe what you did during the lab. How you wired up the board, what code you wrote (include snippets of the code you wrote; the rest of the code should be in the appendix), etc… I'm not expecting an extremely verbose retelling of each step of the lab.

Part 1

We first designed the module for SRAM, this includes 8192, 8-bit data data words. We also need to create a testbench to test on the functionality of the SRAM module by instantiating it in the testbench.

Part 2

The second part of the lab is to create a SPI interface module, which creates the registers to receive serial data for a write operation and the state machines and interface to allow data for these registers to be received. We also need to create the registers to output data on a read instruction and test with dummy data.

Part 3

Instantiate the SPI interface and the SRAM modules into the M23A640 module, and test it with the tbserial.v testbench.

Results

Place the results of your experiments and tasks here. Tables, charts, screenshots, etc.. Answer any questions from the lab document here.

Part 1

```
 0, address=    x : RE = x  WE = x Data =   x
 5, address=    0 : RE = 0  WE = 0 Data =   z
10, address=    0 : RE = 0  WE = 1 Data =   5
15, address=    1 : RE = 0  WE = 1 Data =   8
20, address=    0 : RE = 1  WE = 0 Data =   5
25, address=    1 : RE = 0  WE = 0 Data =   z
30, address=    1 : RE = 1  WE = 0 Data =   x
35, address=    1 : RE = 0  WE = 1 Data =   8
40, address=    1 : RE = 0  WE = 1 Data =   6
45, address=    1 : RE = 1  WE = 0 Data =   x
```

This is the simulation result of our SRAM module.

Part 2

```
Beginning write-read test
Passed write-read test
Beginning write-read test 2
Passed write-read test 2
Beginning broken write test
Passed broken write test
Beginning alias test
Passed alias test
***** - ALL TESTS PASSED - *****
```

What problems did you encounter while testing your steps yourself?
When we tested my steps myself, we encountered some problems with the inout setup and the Data output.

• Did any problems arise when demonstrating for the TA? What were they? Explain your thoughts on how/why these test cases escaped your own testing.
When we demonstrated for the TA, we encountered some problems with our io in the SPI module. This was caused by some issue with the dreg that was assigned to the io. When we demonstrate our SRAM module to the TA, we also have some problems with the inout and the Data output.

Conclusion

In conclusion, this lab provided valuable experience in designing and testing a bidirectional SPI SRAM chip model. We have learned how to design SRAM modules and testbench in verilog, we also got hands-on experience with the Serial Peripheral Interface (SPI), a way for devices to talk to each other using four wires. Overall, the lab enhanced our comprehension of chip models and their applications in real-world scenarios.

Appendix

```
module SRAM (
	input [12:0] Address,
	input RE,
	input WE,
	inout [7:0] Data
);

	reg [7:0] RAM [0:8191];
	reg read_flag;

	assign Data = (read_flag) ? RAM[Address] : 8'bZ;

	always @(posedge WE) begin
		RAM[Address] <= Data;
	end

	always @(posedge RE) begin
		read_flag <= 1;
	end

	always @(negedge RE) begin
		read_flag <= 0;
	end

endmodule
```

```
module tbSram();

  reg [12:0] address;
  reg RE;
  reg WE;
  wire [7:0] Data;
  reg [7:0]DataIn;
  SRAM uut (
		address,
		RE,
		WE,
		Data
```

```
  );
assign Data = DataIn;
initial begin
  $monitor("# %t, address= %d : RE = %d  WE = %d Data = %d",
$time, address,
        RE,
        WE,
        Data);
        #5;
        address = 13'd0;
        DataIn = 8'bZ;
        WE <= 1'b0;
        RE <= 1'b0;

        #5;
        address = 13'd0;
        DataIn = 8'd5;
        WE <= 1'b1;
        RE <= 1'b0;


        #5;
        address = 13'd1;
        DataIn = 8'd8;
        WE <= 1'b1;
        RE <= 1'b0;

        #5;
        address = 13'd0;
        DataIn = 8'bZ;
        WE <= 1'b0;
        RE <= 1'b1;
        #5;
        address = 13'd1;
        DataIn = 8'bZ;
        WE <= 1'b0;
        RE <= 1'b0;
        #5;
        address = 13'd1;
        DataIn = 8'bZ;
        WE <= 1'b0;
        RE <= 1'b1;

        #5;
```

```
        address = 13'd1;
        DataIn = 8'd8;
        WE <= 1'b1;
        RE <= 1'b0;


        #5;
        address = 13'd1;
        DataIn = 8'd6;
        WE <= 1'b1;
        RE <= 1'b0;

        #5;
        address = 13'd1;
        DataIn = 8'bZ;
        WE <= 1'b0;
        RE <= 1'b1;
        #5;


        $finish;
        end

endmodule


module spi(
   input csb,
   input sck,
   input si,
   output reg so,
   inout [7:0] io,
   output reg re,
   output reg we,
   output reg [15:0] address

);


   reg [7:0] inst;
   reg [7:0] dreg;

   reg [5:0] customState;
```

```
    reg [7:0] readreg;
    reg readop; // the signal to check the read operation


    /*
    reg [7:0] io_temp;
    wire [7:0] io_recv;

    assign io= io_temp;
    assign io_recv = io;
*/
    assign io[7:0] = (inst == 8'b00000010) ? dreg[7:0] : 8'bZZZZZZZZ;




    always @(posedge sck) begin

      // io_temp = 8'b0;

       if (~csb) begin
         case (customState)
           //shift in the instruction bits
           0: inst[7:0] <= {inst[6:0], si};
           1: inst[7:0] <= {inst[6:0], si};
           2: inst[7:0] <= {inst[6:0], si};
           3: inst[7:0] <= {inst[6:0], si};
           4: inst[7:0] <= {inst[6:0], si};
           5: inst[7:0] <= {inst[6:0], si};
           6: inst[7:0] <= {inst[6:0], si};
           7: inst[7:0] <= {inst[6:0], si};



           // shift in the address bits
           8: address[15:0] <= {address[14:0], si};
           9: address[15:0] <= {address[14:0], si};
           10: address[15:0] <= {address[14:0], si};
           11: address[15:0] <= {address[14:0], si};
           12: address[15:0] <= {address[14:0], si};
           13: address[15:0] <= {address[14:0], si};
           14: address[15:0] <= {address[14:0], si};
           15: address[15:0] <= {address[14:0], si};
           16: address[15:0] <= {address[14:0], si};
```

```
17: address[15:0] <= {address[14:0], si};
18: address[15:0] <= {address[14:0], si};
19: address[15:0] <= {address[14:0], si};
20: address[15:0] <= {address[14:0], si};
21: address[15:0] <= {address[14:0], si};
22: address[15:0] <= {address[14:0], si};
23: address[15:0] <= {address[14:0], si};


//shift in data for write operations
24: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
25: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
26: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
27: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
28: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
29: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
30: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   end
31: begin
   if (inst == 8'b00000010)
      dreg[7:0] <= {dreg[6:0], si};
   customState <= 0;

   end
```

```
            default: ;
        endcase

    end

    // Determine if it is a read operation based on the lower bits of the address
    readop = (customState == 11) ? (address[1:0] == 0) : readop;

    //if it is read operation then enable read output
    re <= (inst == 8'b00000011) && (customState == 23) && readop ? 1 : 0;

    // check if it is a write operation and enable write output
    we <= (inst == 8'b00000010) && (customState == 31) && (~csb) ? 1 : 0;

    // increment the state machine counter
   customState <= (customState < 33) ? (customState + 1) : customState;

end


always @(negedge sck) begin
if (~csb && (inst== 8'b00000011) && readop) begin //if it operate read
    case (customState)
        24: begin
            so <= io[7];
            readreg <= {io[6:0], 1'b0};
        end

        25: begin
            so <= readreg[7];
            readreg[7:0] <= {readreg[6:0], 1'b0};
        end
        26: begin
            so <= readreg[7];
            readreg[7:0] <= {readreg[6:0], 1'b0};
        end
        27: begin
            so <= readreg[7];
            readreg[7:0] <= {readreg[6:0], 1'b0};
        end
        28: begin
            so <= readreg[7];
```

```
            readreg[7:0] <= {readreg[6:0], 1'b0};
         end
         29: begin
            so <= readreg[7];
            readreg[7:0] <= {readreg[6:0], 1'b0};
         end
         30: begin
            so <= readreg[7];
            readreg[7:0] <= {readreg[6:0], 1'b0};
         end
         31: begin
            so <= readreg[7];
            readreg[7:0] <= {readreg[6:0], 1'b0};
            customState <= 0;
         end

         default: so <= 1'bz;
      endcase
   end
end



always @(negedge csb)
  begin
    customState <= 1'b0;

    so <= 1'bZ;
  end


endmodule


module M23A640(csb, so, holdb, sck, si);
input csb;
output so;
input holdb;
input sck;
input si;

wire [7:0]dreg;
wire WE;
wire RE;
```

```
wire [15:0] address;


spi spi1(csb, sck, si, so, dreg, RE, WE, address);
SRAM sram({address[12:0]}, RE, WE, dreg);

endmodule



module testbench();


reg csb, sck, si;
wire so;
reg [7:0] dreg;
reg fail;
reg anyfail;



M23A640 dut(csb, so, 1'b1, sck, si);

initial begin
  anyfail = 0;
  si = 0;
  csb = 1;
  sck = 1; #5 sck = 0;
  write_read_test();
  write_read_test2();
  broken_write_test();
  alias_test();
  #5 sck = 1; #5 sck = 0;
  #5 sck = 1; #5 sck = 0;
  #5 sck = 1; #5 sck = 0;
  if (anyfail) $display("XXXXX - Failed to pass test suite - XXXXX");
  else $display("***** - ALL TESTS PASSED - *****");
$finish;
end


task write_read_test();
  begin
  // Perform 10 writes then 10 reads to see if the data is read back correctly
```

```
    $display("Beginning write-read test");
   // $monitor("dreg = %h, @ %d", dreg,$time);
    fail = 0;
    write(16'h0001, 8'h01);
    write(16'h0002, 8'h12);
    write(16'h0003, 8'h13);
    write(16'h0003, 8'h23);
    write(16'h0004, 8'h34);
    write(16'h0005, 8'h45);
    write(16'h1001, 8'h56);
    write(16'h1002, 8'h67);
    write(16'h1003, 8'h78);
    write(16'h1004, 8'h89);
    write(16'h1005, 8'h9A);
    read(16'h0001, dreg);  if (dreg !== 8'h01) fail = 1;
    read(16'h0002, dreg);  if (dreg !== 8'h12) fail = 1;
    read(16'h0003, dreg);  if (dreg !== 8'h23) fail = 1;
    read(16'h0004, dreg);  if (dreg !== 8'h34) fail = 1;
    read(16'h0005, dreg);  if (dreg !== 8'h45) fail = 1;
    read(16'h1001, dreg);  if (dreg !== 8'h56) fail = 1;
    read(16'h1002, dreg);  if (dreg !== 8'h67) fail = 1;
    read(16'h1003, dreg);  if (dreg !== 8'h78) fail = 1;
    read(16'h1004, dreg);  if (dreg !== 8'h89) fail = 1;
    read(16'h1005, dreg);  if (dreg !== 8'h9A) fail = 1;
    if (fail == 1) begin
      anyfail = 1;
      $display("XXXXX - Failed write-read test - XXXXX");
    end
    else $display("Passed write-read test");
  end
endtask

task write_read_test2();
  begin
  // Perform some re-writes to if the data is read back correctly
    $display("Beginning write-read test 2");
//    $monitor("dreg = %h, @ %d", dreg,$time);
    fail = 0;
    write(16'h0001, 8'h01);
    write(16'h0002, 8'h12);
    write(16'h0003, 8'h23);
    write(16'h0004, 8'h34);
    write(16'h0005, 8'h45);
    read(16'h0001, dreg);  if (dreg !== 8'h01) fail = 1;
```

```
    read(16'h0002, dreg);  if (dreg !== 8'h12) fail = 1;
    read(16'h0003, dreg);  if (dreg !== 8'h23) fail = 1;
    read(16'h0004, dreg);  if (dreg !== 8'h34) fail = 1;
    read(16'h0005, dreg);  if (dreg !== 8'h45) fail = 1;
    write(16'h0001, 8'h10);
    write(16'h0002, 8'h21);
    write(16'h0003, 8'h32);
    read(16'h0001, dreg);  if (dreg !== 8'h10) fail = 1;
    read(16'h0002, dreg);  if (dreg !== 8'h21) fail = 1;
    read(16'h0003, dreg);  if (dreg !== 8'h32) fail = 1;
    read(16'h0004, dreg);  if (dreg !== 8'h34) fail = 1;
    read(16'h0005, dreg);  if (dreg !== 8'h45) fail = 1;
    if (fail == 1) begin
      anyfail = 1;
      $display("XXXXX - Failed write-read test 2 - XXXXX");
    end
    else $display("Passed write-read test 2");
  end
endtask

task broken_write_test();
  begin
// Test response to a broken write
    $display("Beginning broken write test");
    fail = 0;
    write(16'h0001, 8'h01);
    write(16'h0002, 8'h12);
    read(16'h0001, dreg);  if (dreg !== 8'h01) fail = 1;
    read(16'h0002, dreg);  if (dreg !== 8'h12) fail = 1;
    write(16'h0001, 8'h34);
    broken_write(16'h0002, 8'h56);
    write(16'h0003, 8'h78);
    read(16'h0001, dreg);  if (dreg !== 8'h34) fail = 1;
    read(16'h0002, dreg);  if (dreg !== 8'h12) fail = 1;
    read(16'h0003, dreg);  if (dreg !== 8'h78) fail = 1;
    if (fail == 1) begin
      anyfail = 1;
      $display("XXXXX - Failed broken write test - XXXXX");
    end
    else $display("Passed broken write test");
  end
endtask

task alias_test();
```

```
  begin
// Test alias behavior
    $display("Beginning alias test");
    fail = 0;
    write(16'h0001, 8'h01);
    write(16'h0002, 8'h12);
    read(16'h0001, dreg);  if (dreg !== 8'h01) fail = 1;
    read(16'h0002, dreg);  if (dreg !== 8'h12) fail = 1;
    read(16'h8001, dreg);  if (dreg !== 8'h01) fail = 1;
    read(16'h8002, dreg);  if (dreg !== 8'h12) fail = 1;
    read(16'h4002, dreg);  if (dreg !== 8'hzz) fail = 1;
    write(16'h4001, 8'h34);
    write(16'h4002, 8'h56);
    read(16'h0001, dreg);  if (dreg !== 8'h34) fail = 1;
    read(16'h0002, dreg);  if (dreg !== 8'h56) fail = 1;
    read(16'h8001, dreg);  if (dreg !== 8'h34) fail = 1;
    read(16'h8002, dreg);  if (dreg !== 8'h56) fail = 1;
    read(16'h4002, dreg);  if (dreg !== 8'hzz) fail = 1;
     if (fail == 1) begin
      anyfail = 1;
      $display("XXXXX - Failed alias test - XXXXX");
    end
    else $display("Passed alias test");
  end
endtask

task write(input [15:0] address, input [7:0] data);
  begin
    csb = 0;                                    // Start Transaction
    si = 0; #5 sck = 1;              // Send write instruction 0000 0010
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 1; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;

    #5 si = address[15]; sck = 0; #5 sck = 1;   // Send address MSB first
    #5 si = address[14]; sck = 0; #5 sck = 1;
    #5 si = address[13]; sck = 0; #5 sck = 1;
    #5 si = address[12]; sck = 0; #5 sck = 1;
    #5 si = address[11]; sck = 0; #5 sck = 1;
    #5 si = address[10]; sck = 0; #5 sck = 1;
```

```
    #5 si = address[9]; sck = 0; #5 sck = 1;
    #5 si = address[8]; sck = 0; #5 sck = 1;
    #5 si = address[7]; sck = 0; #5 sck = 1;
    #5 si = address[6]; sck = 0; #5 sck = 1;
    #5 si = address[5]; sck = 0; #5 sck = 1;
    #5 si = address[4]; sck = 0; #5 sck = 1;
    #5 si = address[3]; sck = 0; #5 sck = 1;
    #5 si = address[2]; sck = 0; #5 sck = 1;
    #5 si = address[1]; sck = 0; #5 sck = 1;
    #5 si = address[0]; sck = 0; #5 sck = 1;


    #5 si = data[7]; sck = 0; #5 sck = 1;          // Send data MSB first
    #5 si = data[6]; sck = 0; #5 sck = 1;
    #5 si = data[5]; sck = 0; #5 sck = 1;
    #5 si = data[4]; sck = 0; #5 sck = 1;
    #5 si = data[3]; sck = 0; #5 sck = 1;
    #5 si = data[2]; sck = 0; #5 sck = 1;
    #5 si = data[1]; sck = 0; #5 sck = 1;
    #5 si = data[0]; sck = 0; #5 sck = 1;
    #5 sck = 0; #5 csb = 1; #5;
  end
endtask


task read(input [15:0] address, output reg [7:0] data);
  begin
    csb = 0;                                  // Start Transaction
    si = 0; #5 sck = 1;              // Send read instruction 0000 0011
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 1; sck = 0; #5 sck = 1;
    #5 si = 1; sck = 0; #5 sck = 1;

    #5 si = address[15]; sck = 0; #5 sck = 1;   // Send address MSB first
    #5 si = address[14]; sck = 0; #5 sck = 1;
    #5 si = address[13]; sck = 0; #5 sck = 1;
    #5 si = address[12]; sck = 0; #5 sck = 1;
    #5 si = address[11]; sck = 0; #5 sck = 1;
    #5 si = address[10]; sck = 0; #5 sck = 1;
    #5 si = address[9]; sck = 0; #5 sck = 1;
```

```
    #5 si = address[8]; sck = 0; #5 sck = 1;
    #5 si = address[7]; sck = 0; #5 sck = 1;
    #5 si = address[6]; sck = 0; #5 sck = 1;
    #5 si = address[5]; sck = 0; #5 sck = 1;
    #5 si = address[4]; sck = 0; #5 sck = 1;
    #5 si = address[3]; sck = 0; #5 sck = 1;
    #5 si = address[2]; sck = 0; #5 sck = 1;
    #5 si = address[1]; sck = 0; #5 sck = 1;
    #5 si = address[0]; sck = 0; #5 sck = 1;


    #5 sck = 0; #5 data[7] = so; sck = 1;          // Receive data MSB first
    #5 sck = 0; #5 data[6] = so; sck = 1;
    #5 sck = 0; #5 data[5] = so; sck = 1;
    #5 sck = 0; #5 data[4] = so; sck = 1;
    #5 sck = 0; #5 data[3] = so; sck = 1;
    #5 sck = 0; #5 data[2] = so; sck = 1;
    #5 sck = 0; #5 data[1] = so; sck = 1;
    #5 sck = 0; #5 data[0] = so; sck = 1;
    #5 sck = 0; #5 csb = 1; #5;
//  $display("performing read from %h, data is %h",address, data);
  end
endtask

task broken_write(input [15:0] address, input [7:0] data);
  begin
    csb = 0;                                    // Start Transaction
    si = 0; #5 sck = 1;               // Send write instruction 0000 0010
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;
    #5 si = 1; sck = 0; #5 sck = 1;
    #5 si = 0; sck = 0; #5 sck = 1;

    #5 si = address[15]; sck = 0; #5 sck = 1;   // Send address MSB first
    #5 si = address[14]; sck = 0; #5 sck = 1;
    #5 si = address[13]; sck = 0; #5 sck = 1;
    #5 si = address[12]; sck = 0; #5 sck = 1;
    #5 si = address[11]; sck = 0; #5 sck = 1;
    #5 si = address[10]; sck = 0; #5 sck = 1;
    #5 si = address[9]; sck = 0; #5 sck = 1;
    #5 si = address[8]; sck = 0; #5 sck = 1;
```

```
    #5 si = address[7]; sck = 0; #5 sck = 1;
    #5 si = address[6]; sck = 0; #5 sck = 1;
    #5 si = address[5]; sck = 0; #5 sck = 1;
    #5 si = address[4]; sck = 0; #5 sck = 1;
    #5 si = address[3]; sck = 0; #5 sck = 1;
    #5 si = address[2]; sck = 0; #5 sck = 1;
    #5 si = address[1]; sck = 0; #5 sck = 1;
    #5 si = address[0]; sck = 0; #5 sck = 1;


    #5 si = data[7]; sck = 0; #5 sck = 1;          // Send data MSB first
    #5 si = data[6]; sck = 0; #5 sck = 1;
    #5 si = data[5]; sck = 0; #5 sck = 1;
    #5 si = data[4]; sck = 0; #5 sck = 1;
    #5 si = data[3]; sck = 0; #5 sck = 1;
    #5 si = data[2]; sck = 0; #5 sck = 1;
    #5 si = data[1]; sck = 0; #5 sck = 1;csb = 1;
    #5 si = data[0]; sck = 0; #5 sck = 1;
    #5 sck = 0; #5 ; #5;
  end
endtask
endmodule
```