University Of Waterloo

ECE 429

Task 1
Reading SRECs and Memory

Stephan Van Den Heuval	sjvanden
Ravil Baizhiyenov	rbaizhiy
David Janssen	dajjanss
Ben Ridder	brridder

May 16, 2012

1 Output

Listing 1: Bubble Sort Output

```
# opened file srec_files/BubbleSort.srec
#
# Initializing memory
# read in file
#
# 80020000: 27bdffc8
# 80020004: afbf0034
# 80020008: afbe0030
# 8002000c: 03a0f021
# 80020010: 2402000c
# 80020014: afc20010
# 80020018: 24020009
# 8002001c: afc20014
# 80020020: 24020004
# 80020024: afc20018
# 80020028: 24020063
# 8002002c: afc2001c
# 80020030: 24020078
# 80020034: afc20020
# 80020038: 24020001
# 8002003c: afc20024
# 80020040: 24020003
# 80020044: afc20028
# 80020048: 2402000a
# 8002004c: afc2002c
# 80020050: 27c20010
# 80020054: 00402021
# 80020058: 24050008
# 8002005c: 0c008020
# 80020060: 00000000
# 80020064: 00001021
# 80020068: 03c0e821
# 8002006c: 8fbf0034
# 80020070: 8fbe0030
# 80020074: 27bd0038
# 80020078: 03e00008
# 8002007c: 00000000
# 80020080: 27bdffe8
# 80020084: afbe0014
# 80020088: 03a0f021
# 8002008c: afc40018
# 80020090: afc5001c
# 80020094: afc00000
# 80020098: 0800805f
# 8002009c: 00000000
# 800200a0: 24020001
# 800200a4: afc20004
# 800200a8: 08008055
```

```
# 800200ac: 00000000
# 800200b0: 8fc20004
# 800200b4: 2442ffff
# 800200b8: 00021080
# 800200bc: 8fc30018
# 800200c0: 00621021
# 800200c4: 8c430000
# 800200c8: 8fc20004
# 800200cc: 00021080
# 800200d0: 8fc40018
# 800200d4: 00821021
# 800200d8: 8c420000
# 800200dc: 0043102a
# 800200e0: 10400019
# 800200e4: 00000000
# 800200e8: 8fc20004
# 800200ec: 2442ffff
# 800200f0: 00021080
# 800200f4: 8fc30018
# 800200f8: 00621021
# 800200fc: 8c420000
# 80020100: afc20008
# 80020104: 8fc20004
# 80020108: 2442ffff
# 8002010c: 00021080
# 80020110: 8fc30018
# 80020114: 00621021
# 80020118: 8fc30004
# 8002011c: 00031880
# 80020120: 8fc40018
# 80020124: 00831821
# 80020128: 8c630000
# 8002012c: ac430000
# 80020130: 8fc20004
# 80020134: 00021080
# 80020138: 8fc30018
# 8002013c: 00621021
# 80020140: 8fc30008
# 80020144: ac430000
# 80020148: 8fc20004
# 8002014c: 24420001
# 80020150: afc20004
# 80020154: 8fc3001c
# 80020158: 8fc20000
# 8002015c: 00621823
# 80020160: 8fc20004
# 80020164: 0043102a
# 80020168: 1440ffd1
# 8002016c: 00000000
# 80020170: 8fc20000
# 80020174: 24420001
```

```
# 80020178: afc20000
# 8002017c: 8fc30000
# 80020180: 8fc2001c
# 80020184: 0062102a
# 80020188: 1440ffc5
# 8002018c: 00000000
# 80020190: 03c0e821
# 80020194: 8fbe0014
# 80020198: 27bd0018
# 8002019c: 03e00008
# 800201a0: 00000000
```

Listing 2: Simple Add Output

```
# opened file srec_files/SimpleAdd.srec
#
# Initializing memory
# read in file
#
# 80020000: 27bdffe8
# 80020004: afbe0014
# 80020008: 03a0f021
# 8002000c: 24020003
# 80020010: afc20000
# 80020014: 24020002
# 80020018: afc20004
# 8002001c: afc00008
# 80020020: 8fc30000
# 80020024: 8fc20004
# 80020028: 00621021
# 8002002c: afc20008
# 80020030: 8fc20008
# 80020034: 03c0e821
# 80020038: 8fbe0014
# 8002003c: 27bd0018
# 80020040: 03e00008
# 80020044: 00000000
```

Listing 3: Simple If Output

```
# opened file srec_files/SimpleIf.srec
#
# Initializing memory
# read in file
#
# 80020000: 27bdffe8
# 80020004: afbe0014
# 80020008: 03a0f021
# 8002000c: 24020003
# 80020010: afc20000
# 80020014: 24020002
# 80020018: afc20004
```

```
# 8002001c: afc00008
# 80020020: 8fc20000
# 80020024: 28420005
# 80020028: 10400007
# 8002002c: 00000000
# 80020030: 8fc30000
# 80020034: 8fc20004
# 80020038: 00621021
# 8002003c: afc20000
# 80020040: 08008016
# 80020044: 00000000
# 80020048: 8fc30000
# 8002004c: 8fc20004
# 80020050: 00621023
# 80020054: afc20000
# 80020058: 8fc30000
# 8002005c: 8fc20004
# 80020060: 00621021
# 80020064: afc20008
# 80020068: 8fc20008
# 8002006c: 03c0e821
# 80020070: 8fbe0014
# 80020074: 27bd0018
# 80020078: 03e00008
# 8002007c: 00000000
```

2 Source Code

Listing 4: Single Port Memory Block

```
//
// mem.v
//
// Single port memory. Supports reads and writes.
//
// Write: wren = '1'
// Read: wren = '0'
//
// Big endian
//

module mem(
          clock,
          address,
          wren,
       data_in,
          data_out
);

    // Parameters
```

```
parameter MEM_DEPTH=1048578; // 1 MB of memory
parameter MEM_WIDTH=8; // 8 bit = 1 byte

// Inputs
input wire clock;
input wire[0:31] address;
input wire wren;
input wire[0:31] data_in;

// Outputs
output reg[0:31] data_out;

// Internals
integer i;
wire[0:31] data; // Temperory data storage
reg[0:MEM_WIDTH-1] ram[0:MEM_DEPTH-1]; // The memory

// Set the memory into a known inital state
initial
begin
    $display("Initializing memory");
    for (i = 0; i < MEM_DEPTH; i = i+1) begin
        ram[i] = 0;
    end
end

// Data is put on a mux to only set on the reads
assign data = !wren ? {ram[address], ram[address+1],
                    ram[address+2], ram[address+3]} : 32'h0000_0000;

// Rising edge, load the data
always @(posedge clock)
begin
    if (address < MEM_DEPTH) begin
        if (wren == 1'b1) begin
            fork
            ram[address] <= data_in[0:7];
            ram[address+1] <= data_in[7:15];
            ram[address+2] <= data_in[16:23];
            ram[address+3] <= data_in[24:31];
            join
        end
    end
end

// Falling edge, set the output data
always @(negedge clock)
begin
    if (address < MEM_DEPTH) begin
        data_out = data;
    end else begin
```

```
            data_out = 32'h0000_0000;
        end
    end
endmodule
```

Listing 5: Memory Test Bench

```
//
// mem_tb.v
//
// Memory test bench
//

module mem_tb;
    reg clock, wren;
    reg[0:31] address, data;
    wire[0:31] data_out;

    // Instantiate mem DUT
    mem DUT(clock, address, wren, data, data_out);

    // Executes only once during the initialization
    initial begin
        clock = 0;
        wren = 0;
        address = 0;
        data = 0;
    end

    // Clock generation
    always
        #5 clock = !clock;


    // Setup for simulation
    initial begin
        $dumpfile("mem.vcd"); // waveform
        $dumpvars; // dump all the signals
    end

    initial begin
        // print header to stdout
        $display("\t\ttime,\tclock,\twren,\taddress,\tdata,\tdata_out");
        // prints signal values when they change
        $monitor("%d,\t%b,\t%b,\t%h,\t%h,\t%h", $time, clock, wren, address, data, data_out);
    end

    // Specify when to stop the simulation
    event terminate_sim;
    initial begin
        @ (terminate_sim);
        #10 $finish;
```

```
end

// TEST CASES BEGIN HERE
initial begin
    // 1. load and validate data at mem top and bottom
    @ (posedge clock);
    wren = 1'b1;
    address = 32'h0000_0000;
    data = 32'h55cc_55cc;
    @ (posedge clock);
    wren = 1'b0;
    @ (posedge clock);
    if (data_out != 32'h55cc_55cc) begin
        $display("Mem error at time %d", $time);
        $display("Expected value 55cc55cc, actual value %h", data_out);
        $display("Address %d", address);
        #5 -> terminate_sim;
    end else begin
        $display("Store at the start of mem block - OK");
    end

    @ (posedge clock);
    wren = 1'b1;
    address = 32'h000f_fffc;
    data = 32'h1234_5678;
    @ (posedge clock);
    wren = 1'b0;
    @ (posedge clock);
    if (data_out != 32'h1234_5678) begin
        $display("Mem error at time %d", $time);
        $display("Expected value 12345678, actual value %d", data_out);
        $display("Address %d", address);
        #5 -> terminate_sim;
    end else begin
        $display("Store at the end of mem block - OK");
    end

    // 2. Load and validate data somewhere in the middle of memory
    @ (posedge clock);
    wren = 1'b1;
    address = 32'h0008_8888;
    data = 32'hfedc_ba98;
    @ (posedge clock);
    wren = 1'b0;
    @ (posedge clock);
    if (data_out != 32'hfedc_ba98) begin
        $display("Mem error at time %d", $time);
        $display("Expected value fedcba98, actual value %d", data_out);
        $display("Address %d", address);
        #5 -> terminate_sim;
    end else begin
```

```
            $display("Store in the middle of mem block - OK");
        end

        // 3. Load and validate data just out of memory bounds
        @ (posedge clock);
        wren = 1'b1;
        address = 32'h0010_0000;
        data = 32'h0000_0000;
        @ (posedge clock);
        wren = 1'b0;
        @ (posedge clock);
        if (data_out != 32'h0000_0000) begin
            $display("Mem error at time %d", $time);
            $display("Expected value 00000000, actual value %d", data_out);
            $display("Address %d (out of bounds)", address);
            #5 -> terminate_sim;
        end else begin
            $display("Test store just out of mem bounds - OK");
        end

        // 4. Load and validate data far out of memory bounds
        @ (posedge clock);
        wren = 1'b1;
        address = 32'hffff_ffff;
        data = 32'hb00d_ecaf;
        @ (posedge clock);
        wren = 1'b0;
        @ (posedge clock);
        if (data_out != 32'h0000_0000) begin
            $display("Mem error at time %d", $time);
            $display("Expected value 00000000, actual value %d", data_out);
            $display("Address %d (out of bounds)", address);
            #5 -> terminate_sim;
        end else begin
            $display("Test store far out of mem bounds - OK");
        end
        -> terminate_sim;
    end
endmodule
```

Listing 6: Memory Controller Unit

```
//
// mem_controller.v
//
// Memory Controller Unit.
//
// Currently, it only converts the program code addresses (logical addresses)
// into physical addresses.
//
// In the future, it will be able to load half words from memory appropriately
//
```

```
// Consequently, if we don't need to be able to load half words, then this
// could be compacted into the mem.v
//


module mem_controller(
    clock,
    address,
    wren,
    data_in,
    data_out
);

// Parameters
parameter START_ADDRESS = 32'h80020000;
parameter MAX_ADDRESS = 32'hFFFF_FFFF;

// Ports are just wires to pass straight through

// Inputs
input wire clock;
input wire[0:31] address;
input wire wren;
input wire[0:31] data_in;

// Outputs
output wire[0:31] data_out;

wire[0:31] mem_address;
// Match the logical address to the physical address
assign mem_address = (address < START_ADDRESS) ? MAX_ADDRESS : address - START_ADDRESS;

mem memory0(
    .clock (clock),
    .address (mem_address),
    .wren (wren),
    .data_in (data_in),
    .data_out (data_out)
    );

endmodule
```

Listing 7: SREC Parser

```
//
// srec_parser.v
//
// This module is responsible for taking a given srec file and decoding it.
// Once decoded, it is loaded into memory using the mem_controller module.
//

module srec_parser(
```

```
                clock,
                mem_address,
                mem_wren,
                mem_data_in,
                mem_data_out,
                done,
                bytes_read
);

    parameter SREC_FILE_NAME = "test2.srec";

    input wire clock;

    // Connections to the memory bus
    output reg[0:31] mem_address;
    output reg mem_wren;
    output reg[0:31] mem_data_in;
    input wire[0:31] mem_data_out;

    output reg done;
    output integer bytes_read;

    integer file;
    reg[47*8-1:0] srec_line;
    reg[0:7]  first_character;
    reg[0:7]  second_character;
    integer  length;
    reg [36*8-1:0]  data;
    integer   data_length;
    integer   address;
    integer   data_word;
    integer   offset;

    initial begin
        done = 1'b0;
        bytes_read = 0;

        file = $fopen(SREC_FILE_NAME, "r");
        $display("opened file %s\n", SREC_FILE_NAME);
        length = 1;
        while (length != 0) begin
               length = $fgets(srec_line, file);
               $sscanf(srec_line, "%c%c", first_character, second_character);

               if (second_character == "3") begin
                   $sscanf(srec_line, "S3%2X%8X%s", data_length, address, data);
                   offset = 2*8*(data_length - 4) -64;
                   address = address - 4;

                   @(posedge clock);
```

```
                while (offset >= 16) begin
                    address = address + 4;
                    bytes_read = bytes_read + 4;
                    $sscanf(data[offset+:64], "%8X", data_word);
                    offset = offset - 64;
                    @(posedge clock);
                    mem_address = address;
                    mem_wren = 1'b1;
                    mem_data_in = data_word;
                end
            end
        end
        $display("read in file \n");
        done = 1'b1;
    end
endmodule
```

Listing 8: SREC Parser Test Bench

```
//
// srec_parser_tb.v
//
// Srec Parser Unit Test Bench
//

module srec_parser_tb();
    reg clock;

    wire[0:31] srec_address;
    wire srec_wren;
    wire[0:31] srec_data_in;
    wire[0:31] srec_data_out;

    wire[0:31] address;
    wire wren;
    wire[0:31] data_in;
    wire[0:31] data_out;

    reg[0:31] tb_address;
    reg tb_wren;
    reg[0:31] tb_data_in;
    wire[0:31] tb_data_out;

    wire srec_done;
    wire[0:31] bytes_read;
    integer byte_count;
    integer read_word;

    // Mux the memory bus between the srec and the testbench
    assign address = srec_done ? tb_address : srec_address;
    assign wren = srec_done ? tb_wren : srec_wren;
    assign tb_data_out = data_out;
```

```
    assign data_in = srec_done ? tb_data_in : srec_data_in;

    srec_parser #("srec_files/BubbleSort.srec") U0(
            .clock (clock),
            .mem_address (srec_address),
            .mem_wren (srec_wren),
            .mem_data_in (srec_data_in),
            .mem_data_out (srec_data_out),
            .done (srec_done),
            .bytes_read(bytes_read)
    );

    mem_controller U1(
            .clock (clock),
                    .address (address),
                    .wren (wren),
                    .data_in (data_in),
                    .data_out (data_out)
    );

   initial begin
      clock = 1;
   end

   always begin
    #5 clock = !clock;
   end

   initial begin
       @(posedge srec_done);
       @(posedge clock);

       byte_count = bytes_read;
       tb_address = 32'h8002_0000;
       tb_wren = 1'b0;
       while (byte_count > 0) begin
           @(posedge clock);
               read_word = tb_data_out;

               $display("%8X: %8X", tb_address, read_word);
               tb_address = tb_address + 4;
               byte_count = byte_count - 4;
       end
    end
endmodule // srec_parser_tb
```