Contents

[ECE429\_Memory.v: Memory Module 2](#_Toc388905286)

[ECE429\_SRECParser.v: SREC Parser Module 3](#_Toc388905287)

[ECE429\_Memory\_tb.v: Simple Memory Test Bench 8](#_Toc388905288)

[ECE429\_SRECParser\_tb.v: SREC Parser and Memory Test Bench 9](#_Toc388905289)

# ECE429\_Memory.v: Memory Module

`define MEMORY\_SIZE\_BYTES 1048576 // 1 MB

module ECE429\_Memory(clock, address, datain, dataout, access\_size, r\_w);

input clock;

input [0:31] address;

input [0:31] datain;

input [0:1] access\_size; // 11 for word, 10 for half-word, 01/00 byte

input r\_w; // 0 to read, 1 to write

output [0:31] dataout;

wire[0:31] mod\_addr;

reg[0:31] tmp\_address;

reg[0:31] tmp\_data;

reg[0:1] tmp\_access\_size;

reg tmp\_r\_w;

reg[0:7] memory[0: `MEMORY\_SIZE\_BYTES -1 ];

assign dataout = (!tmp\_r\_w & !clock) ? tmp\_data : 32'bz;

assign mod\_addr = address - 32'h80020000;

always @ (negedge clock)

begin

if( tmp\_r\_w ) begin

if (tmp\_access\_size[0] == 0) begin //byte

memory[tmp\_address] = tmp\_data[24:31];

end else if (tmp\_access\_size[1] == 0) begin //half-word

memory[tmp\_address] = tmp\_data[16:23];

memory[tmp\_address + 1] = tmp\_data[24:31];

end else begin //word

memory[tmp\_address] = tmp\_data[0:7];

memory[tmp\_address + 1] = tmp\_data[8:15];

memory[tmp\_address + 2] = tmp\_data[16:23];

memory[tmp\_address + 3] = tmp\_data[24:31];

end

end

//$display("\t\t%h\t%h\t%h\t%h\t%h\t%h\t%h\t%h\t%h", clock, address, tmp\_address, datain, tmp\_data, access\_size, tmp\_access\_size, tmp\_r\_w, dataout);

end

always @ (posedge clock)

begin

tmp\_access\_size = access\_size;

tmp\_r\_w = r\_w;

tmp\_address = mod\_addr;

if ( !tmp\_r\_w ) begin

if (tmp\_access\_size[0] == 0) begin //byte

tmp\_data = {24'h000000, memory[tmp\_address]};

end else if (tmp\_access\_size[1] == 0) begin //half-word

tmp\_data = {16'h0000, memory[tmp\_address], memory[tmp\_address + 1]};

end else begin //word

tmp\_data = {memory[tmp\_address], memory[tmp\_address + 1], memory[tmp\_address + 2], memory[tmp\_address + 3]};

end

end else begin

tmp\_data = datain;

end

//$display("\t\t%h\t%h\t%h\t%h\t%h\t%h\t%h\t%h\t%h", clock, address, tmp\_address, datain, tmp\_data, access\_size, tmp\_access\_size, tmp\_r\_w, dataout);

end

endmodule

# ECE429\_SRECParser.v: SREC Parser Module

/\*

\* Note: A rising edge of the parseEnable input will cause the parser to run.

\*/

module ECE429\_SRECParser( clock, parseEnable, parseAddr, memData, parseAccessSize, parseDone, parseError );

input clock;

input parseEnable;

output parseAddr;

output memData;

output parseAccessSize;

output parseDone;

output parseError;

reg[0:31] parseAddr; // A 32-bit address to put into the memory

reg[0:31] memData; // A 32-bit piece of data to write to memory

reg[0:1] parseAccessSize; // The access size to write to the memory using

reg parseDone; // Set to 1 once parser is done

reg parseError; // Set to 1 on error

reg done; // Set to 1 once parser is done

reg error; // Set to 1 on error

parameter SRECFileName = "BubbleSort.srec";

parameter MaxSRecordSize = 78; // The maximum size of an S-Record is 78 bytes

integer SRECFile; // The input SREC file

integer lineCount; // A count of the number of characters in the current line of the SREC

reg[0:(MaxSRecordSize\*8)-1] currLine; // Stores the current line from the SREC file (each line is no more than 78 bytes)

reg[0:7] charTypeS; // The first character of the S-Record; this should be "S"

reg[0:7] charTypeCode; // The second character of the S-Record; this should be "0"-"9" (ASCII)

reg[0:(2\*8)-1] count; // Count of how many bytes in address+data+checksum

reg[0:(4\*8)-1] address; // To store the address in the address field of an S-Record (parsed as hex)

reg[0:(MaxSRecordSize\*8)-1] data; // The data in the data field of an S-Record (raw; no parsed into hex yet)

reg[0:7] dataByte; // A byte of data to write to the memory (parsed as hex)

reg[0:7] checksum; // Checksum in S-Record (parsed as hex)

reg[0:7] datacount; // Number of data bytes in the SREC

integer offset;

integer dummyReturn;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ASSIGNMENTS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

always @ (address) begin

parseAddr = address;

end

always @ (dataByte) begin

memData = { 24'h000, dataByte };

end

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* TASKS

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// TASK TO PARSE MAIN INFORMATION FROM SREC

task parseSREC;

input [0:7] temp\_type; // Type of SREC (0-9)

input [0:7] temp\_addrChar; // Number of ASCII char for address

input [0:(MaxSRecordSize\*8)-1] temp\_line; // The string holding the SREC

output [0:7] temp\_datacount; // Number of bytes (char pairs) to read from data field

output [0:(4\*8)-1] temp\_address; // Starting address

output [0:(MaxSRecordSize\*8)-1] temp\_data; // Data field

reg[0:7] temp\_addrBytesString;

reg[0:7] temp\_count;

begin

temp\_addrBytesString = temp\_addrChar + 48; // Digit to ASCII number

// Extract the count, address, and data

dummyReturn = $sscanf(temp\_line, { "S", temp\_type, "%2h", "%",temp\_addrBytesString,"h", "%s" }, temp\_count, temp\_address, temp\_data);

//$display("Temp\_data=%s",temp\_data[0:(MaxSRecordSize\*8)-1]);

// Reduce count by 1 (checksum) and the number of bytes (char pairs) in the address field, so it now counts the data bytes

temp\_datacount = temp\_count - 1 - (temp\_addrChar/2);

end

endtask

always @(posedge parseEnable) begin

done = 1'b0;

error = 1'b0;

parseDone = 1'b0;

parseError = 1'b0;

// Default memory parameters. Access size if 00 for byte-addressable.

parseAccessSize = 2'b00;

// Open the SREC file to read. Exit if could not read it

SRECFile = $fopen(SRECFileName,"r");

if (SRECFile == 0) begin

$display("Problem opening SREC file.\n");

error = 1'b1;

end else begin

// Sequentially parse each line in the SREC file

while ( !done && !error ) begin

// Read the next line. Reached the end if count

lineCount = $fgets(currLine,SRECFile);

if( lineCount == 0 ) begin

$display("Done parsing SREC file. Exiting parser.\n");

done = 1'b1;

end else begin

// Check the first two characters on the SREC to get the type.

dummyReturn = $sscanf(currLine, "%c%c", charTypeS, charTypeCode);

// If the line does not start with an "S", error.

if( charTypeS != 8'h53 ) begin

$display("Error: First character of SREC not an 'S'.\n");

error = 1'b1;

end else begin

// Test the type codes and parse accordingly

case( charTypeCode )

8'h30 :

begin // S0

// Parse the SREC

parseSREC( charTypeCode, 4, currLine, datacount, address, data );

end

8'h31 :

begin // S1

// Parse the SREC

parseSREC( charTypeCode, 4, currLine, datacount, address, data );

end

8'h32 :

begin // S2

// Parse the SREC

parseSREC( charTypeCode, 6, currLine, datacount, address, data );

end

8'h33 :

begin // S3

// Parse the SREC

parseSREC( charTypeCode, 8, currLine, datacount, address, data );

end

8'h35 :

begin // S5

// Parse the SREC

parseSREC( charTypeCode, 4, currLine, datacount, address, data );

end

8'h36 :

begin // S6

// Parse the SREC

parseSREC( charTypeCode, 6, currLine, datacount, address, data );

end

8'h37 :

begin // S7

// Parse the SREC

parseSREC( charTypeCode, 8, currLine, datacount, address, data );

// Parser done

done = 1'b1;

end

8'h38 :

begin // S8

// Parse the SREC

parseSREC( charTypeCode, 6, currLine, datacount, address, data );

// Parser done

done = 1'b1;

end

8'h39 :

begin // S9

// Parse the SREC

parseSREC( charTypeCode, 4, currLine, datacount, address, data );

// Parser done

done = 1'b1;

end

default :

begin // Error on any other type codes

$display({"Error: Unrecognised SREC type code: ",charTypeCode,"\n"});

error = 1'b1;

end

endcase

if( error != 1 ) begin

// Display the checksum, address, and data in hex

$display("S%c Record: start\_address=0x%h data=0x", charTypeCode, address);

// Convert the data to hex and display it, byte by byte, starting at the beginning of the valid data. Write to memory

// It is -2 since 2 for checksum

offset = (MaxSRecordSize - 2\*datacount - 2)\*8;

while( datacount > 0 ) begin

// Convert next data byte pair to hex

dummyReturn = $sscanf(data[offset +: (2\*8)], "%2h", dataByte);

$display("%h", dataByte);

// Hold the values steady from before a rising edge and after a falling edge.

if( (charTypeCode == 8'h31) || (charTypeCode == 8'h32) ||(charTypeCode == 8'h33) ) begin

//$display("S3 rec");

@(posedge clock);

@(negedge clock);

end

// Increment offset to look at next byte pair. Decrement datacount so know when to stop.

offset = offset + (2\*8);

datacount = datacount - 1;

// Move the address over to write to next byte in memory

address = address + 1;

end

// Print checksum

dummyReturn = $sscanf(data[offset +: (2\*8)], "%2h", checksum);

$display("checksum=0x%h\n", checksum);

end

end

end

end

end

$display("Done parsing SREC file. Exiting parser.\n");

$fclose(SRECFile); // Close the SREC file

parseDone = done;

parseError = error;

end

endmodule

# ECE429\_Memory\_tb.v: Simple Memory Test Bench

include "ECE429\_Memory.v";

module ECE429\_Memory\_tb();

reg clock;

reg [0:31] address;

reg [0:31] datain;

reg [0:1] access\_size; // 11 for word, 10 for half-word, 01/00 byte

reg r\_w; // 0 to read, 1 to write

wire [0:31] dataout;

initial begin

$display("time\tclock\taddress\tdatain\taccess\_size\tr\_w\tdataout");

$monitor("%g\t%h\t%h\t\t\t\t\%h\t\t\t\t%h\t\t%h\t%h",

$time, clock, address, datain, access\_size, r\_w, dataout);

clock = 0;

address = 32'h80020000;

datain = 32'h10001000;

access\_size = 2'b00;

r\_w = 1;

#600

address = 32'h80020000;

datain = 32'h10001000;

access\_size = 2'b00;

r\_w = 0;

#1600 $finish;

end

always begin

#100 clock = ~clock;

end

always @ (negedge clock) begin

if ($time < 600) begin

access\_size = access\_size + 1'b1;

datain = datain + 1'h1;

address = address + 32'h00010000;

end else begin

access\_size = access\_size + 1'b1;

address = address + 32'h00010000;

end

end

ECE429\_Memory m(

.clock(clock),

.address(address),

.datain(datain),

.access\_size(access\_size),

.r\_w(r\_w),

.dataout(dataout)

);

endmodule

# ECE429\_SRECParser\_tb.v: SREC Parser and Memory Test Bench

include "ECE429\_SRECParser.v";

module ECE429\_SRECParser\_tb();

reg clock;

reg parseEnable;

wire[0:31] parseAddr; // A 32-bit address to put into the memory

wire[0:31] memAddr;

wire[0:31] memData; // A 32-bit piece of data to write to memory

wire[0:1] parseAccessSize; // The access size to write to the memory using

wire[0:1] memAccessSize;

wire memR\_W; // Whether to read or write to memory

wire parseDone; // Set to 1 once parser is parseDone

wire parseError; // Set to 1 on parseError

reg[0:31] readAddr;

reg[0:31] readAccessSize;

reg[0:31] maxReadAddr;

wire [0:31] dataout; // To see output of the memory

initial begin

clock = 0;

maxReadAddr = 32'h00000000;

// Toggle parse enable

parseEnable = 0;

#1

parseEnable = 1;

#1

parseEnable = 0;

readAccessSize = 2'b11;

readAddr = 32'h80020000;

@(posedge clock);

readAddr = parseAddr;

//$monitor("%g\t%b\t%b\t%b", $time, clock, parseDone, parseError);

//$monitor("%b\t%b\n", parseDone, parseError);

end

assign memR\_W = ~parseDone;

assign memAddr = (parseDone) ? readAddr : parseAddr;

assign memAccessSize = (parseDone) ? readAccessSize : parseAccessSize;

always begin

#10 clock = ~clock;

end

always @(parseAddr) begin

if(parseAddr > maxReadAddr) begin

maxReadAddr = parseAddr;

end

end

always @(parseError) begin

if(parseError == 1) begin

$finish;

end

end

always @(negedge clock) begin

if (parseDone == 1) begin

$display("read at mem[0x%h] = 0x%h\n", readAddr, dataout);

readAddr = readAddr + 4;

//if(readAddr == (32'h80020000 + 1000)) begin

if(readAddr >= maxReadAddr) begin

$finish;

end

end

end

ECE429\_SRECParser #("SimpleIf.srec") s(

.clock(clock),

.parseEnable(parseEnable),

.parseAddr(parseAddr),

.memData(memData),

.parseAccessSize(parseAccessSize),

.parseDone(parseDone),

.parseError(parseError)

);

ECE429\_Memory m(

.clock(clock),

.address(memAddr),

.datain(memData),

.access\_size(memAccessSize),

.r\_w(memR\_W),

.dataout(dataout)

);

endmodule