

UNIVERSITY OF WATERLOO AQ  
Faculty of Engineering   
      Department of Electrical and Computer Engineering         
ECE 621- Computer Organization

**Main Memory and SREC Parser**

Group 2

**Jordan *Ross,* Mustafa Faraj**   
j25ross, mabdulaz 26 September, 2014

# Memory Module Code

This section contains the code for the main memory module. The main memory implemented follows the specifications provided; namely, it is byte addressable and allows for 8, 16 or 32 bit read/write requests. Data is stored at an offset of 'h80020000 and the memory size is 1MB by default, configurable.

**module** memory**(**

**output** **reg** **[**31**:**0**]** data\_out**,**

**input** **[**31**:**0**]** address**,**

**input** **[**31**:**0**]** data\_in**,**

**input** write**,**

**input** clk**,**

**input** **[**1**:**0**]** access\_size

**);**

**reg** **[**31**:**0**]** buffer**;**

**parameter** size **=** 'h100000**;**

**parameter** offset **=** 'h80020000**;**

**reg** **[**7**:**0**]** memory **[**0**:**size**];** //1048577 = 1MB

**always** **@(posedge** clk**)** **begin** // on the positive edge we will clear the value of the data\_out line

data\_out **=** 32'h00000000**;**

**end**

// On every negative edge of the clock cycle we will write/read values into memory based the address that was set on the rising edge.

**always** **@(negedge** clk**)** **begin**

**if** **(**write**)** **begin** // write the value from data\_in into memory

**if** **(**access\_size **==** 2'b10**)** **begin** //32 bit access

memory**[**address**-**offset**]** **=** data\_in**[**31**:**24**];**

memory**[(**address**-**offset**)** **+** 1**]** **=** data\_in**[**23**:**16**];**

memory**[(**address**-**offset**)** **+** 2**]** **=** data\_in**[**15**:**8**];**

memory**[(**address**-**offset**)** **+** 3**]** **=** data\_in**[**7**:**0**];**

**end**

**if** **(**access\_size **==** 2'b01**)** **begin** //16 bit access

memory**[(**address**-**offset**)]** **=** data\_in**[**15**:**8**];**

memory**[(**address**-**offset**)** **+** 1**]** **=** data\_in**[**7**:**0**];**

**end**

**if** **(**access\_size **==** 2'b00**)** **begin** //8 bit access

memory**[(**address**-**offset**)]** **=** data\_in**[**7**:**0**];**

**end**

**end**

**else** **begin** // Read the value into data\_out

**if** **(**access\_size **==** 'b10**)** **begin** //32 bit access

data\_out**[**7**:**0**]** **=** memory**[(**address**-**offset**)** **+** 3**];**

data\_out**[**15**:**8**]** **=** memory**[(**address**-**offset**)** **+** 2**];**

data\_out**[**23**:**16**]** **=** memory**[(**address**-**offset**)** **+** 1**];**

data\_out**[**31**:**24**]** **=** memory**[(**address**-**offset**)];**

**end**

**if** **(**access\_size **==** 'b01**)** **begin** //16 bit access

data\_out**[**7**:**0**]** **=** memory**[(**address**-**offset**)** **+** 1**];**

data\_out**[**15**:**8**]** **=** memory**[(**address**-**offset**)];**

**end**

**if** **(**access\_size **==** 'b00**)** **begin** //8 bit access

data\_out**[**7**:**0**]** **=** memory**[(**address**-**offset**)];**

**end**

**end**

**end**

**endmodule**

# Memory Module Test Bench

This section contains the code for a simple test bench that was used to verify the functionality of the memory module. In its current state, the test consists of writing a single 32-bit word into memory at address 32'h80020000, followed by reading the data from the same memory address in 32, 16 and 8-bit chunks.

The verification section, later in this report, shows screenshots of the behaviour of this test bench and demonstrates fictional memory behaviour.

**module** memory\_tb**;**

// Output ports for testing

**wire** **[**31**:**0**]** data\_out**;**

// Input ports for testing

**reg** **[**31**:**0**]** address**;**

**reg** **[**31**:**0**]** data\_in**;**

**reg** write**;**

**reg** clk**;**

**reg** **[**1**:**0**]** access\_size**;**

**integer** fh **=** 0**;** // file handler for output

// Instantiate the memory module

memory memory\_uut**(.**data\_out**(**data\_out**),**

**.**address**(**address**),**

**.**data\_in**(**data\_in**),**

**.**write**(**write**),**

**.**clk**(**clk**),**

**.**access\_size**(**access\_size**)**

**);**

// Parameters to define the address spaces

**parameter** instruction\_offset **=** 32'h80020000**;**

**initial** **begin**

// Start the clock high

clk **=** 1**;**

**#**100**;** // delay 1 clock cycle

// Start giving inputs here

// We want to test writing an value to address 0x80020000 for a program instruction

// Set the address line on the rising edge;

address **=** instruction\_offset**+**0**;**

write **=** 1**;**

**#**50**;** // delay until falling edge

// Set the data line on the falling edge;

data\_in **=** 32'h98765432**;**

access\_size **=** 2'b10**;**

write **=** 1**;**

**#**50**;**

address **=** instruction\_offset**+**0**;**

write **=** 0**;**

access\_size **=** 2'b10**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

**#**50**;**

address **=** instruction\_offset**+**0**;**

write **=** 0**;**

access\_size **=** 2'b01**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

**#**50**;**

address **=** instruction\_offset**+**0**;**

write **=** 0**;**

access\_size **=** 2'b00**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

**#**50**;**

// Now we are going to test writing a 16-bit value and then reading it back.

// We want to test writing an value to address 0x80020008 for a program instruction

// Set the address line on the rising edge;

address **=** instruction\_offset**+**8**;**

write **=** 1**;**

**#**50**;** // delay until falling edge

// Set the data line on the falling edge;

data\_in **=** 16'hAAAA**;**

access\_size **=** 2'b01**;**

**#**50**;**

address **=** instruction\_offset**+**8**;**

write **=** 0**;**

access\_size **=** 2'b01**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

**#**50**;**

// Now we are going to test writing a 8-bit value and then reading it back.

// We want to test writing an value to address 0x8002000D for a program instruction

// Set the address line on the rising edge;

address **=** instruction\_offset**+**12**;**

write **=** 1**;**

**#**50**;** // delay until falling edge

// Set the data line on the falling edge;

data\_in **=** 8'hBB**;**

access\_size **=** 2'b00**;**

**#**50**;**

address **=** instruction\_offset**+**12**;**

write **=** 0**;**

access\_size **=** 2'b00**;**

**#**50**;** // delay until falling edge

// read the data line on the falling edge;

**#**50**;**

**end**

**always** **begin**

**#**50 clk **=** **!**clk**;**

**end**

**endmodule**

# SREC Parser

This section contains an implementation for an SREC parser. Currently this parser is capable of dealing with records of types S1, S2, S3 and S7. This parser also writes the content of the SREC files into the main memory at the specified addresses by the records. The parser starts reading in the file, a character at a time, and terminates when it reaches the EOF character (8'hff).

**module** srec\_parser**;**

// Output ports for memory module

**wire** **[**31**:**0**]** data\_out**;**

// Input ports for memory module

**reg** **[**31**:**0**]** address**;**

**reg** **[**31**:**0**]** data\_in**;**

**reg** write**;**

**reg** clk**;**

**reg** **[**1**:**0**]** access\_size**;**

**integer** fh **=** 0**;** // file handler for output

**integer** i **=** 0**;** // loop variable

**integer** data\_byte **=** 0**;** // variable to keep track of what byte we are on.

**integer** data\_offset **=** 0**;** // keep track of the offset from the data address to write the next byte.

**reg** **[**1**:**0**]**nibble\_count **=** 0**;** // keep track of which nibble is being written (upper/lower).

**reg** **[**7**:**0**]**rec\_type**;** // record type number

**reg** **[**7**:**0**]** byte\_count**;** // the number of bytes for the address, data, and checksum

**integer** record\_code**;** // A record\_code is equivalent to 1 ASCII digit/letter in the .srec file

**reg** **[**31**:**0**]** rec\_address**;** // the address given by the record.

**reg** **[**7**:**0**]**rec\_data**;** // A single byte of the data from the record.

**reg** **[**7**:**0**]** temp**;** // a temporary byte used for place holding.

**reg** done **=** 0**;** // this will set high when we are done parsing the file.

**reg** **[**7**:**0**]** file\_char **=** 8'h0A**;**

// Instantiate the memory module

memory memory**(.**data\_out**(**data\_out**),**

**.**address**(**address**),**

**.**data\_in**(**data\_in**),**

**.**write**(**write**),**

**.**clk**(**clk**),**

**.**access\_size**(**access\_size**)**

**);**

// Parameters to define the address spaces

**parameter** instruction\_offset **=** 32'h80020000**;**

**initial** **begin**

$monitor**(**"Starting the SREC parser..."**);**

// Open the SREC file to read

fh **=** $fopen**(**"D:/GitHub/ECE621\_PiplinedProcessor/BubbleSort.srec"**,** "r"**);**

// Start the clock high

clk **=** 1**;**

// loop until we set the done bit

**while** **(**done **==** 0**)** **begin**

**#**100**;** // Delay 1 clock cycle.

// Read the first/next character from the file.

file\_char **=** $fgetc**(**fh**);**

**if** **(**file\_char **==** 8'hff**)** **begin**

done **=** 1**;**

file\_char **=** 8'h0A**;**

**end**

// Reset the record byte which keeps track of the current byte of the line you are reading in.

// This is equivalent to 1 ASCII code from the file.

record\_code **=** 0**;**

// Loop until we reach a new line character which signifies a new record.

**while** **(**file\_char **!=** 8'h0A**)** **begin**

**#**50**;** // Delay 1/2 clock cycle.

**if** **(**record\_code **==** 0**)** **begin**

// Clear out all the bit fields.

rec\_type **=** 8'h4**;**

byte\_count **=** 16'h0**;**

rec\_address **=** 32'h0**;**

rec\_data **=** 132'h0**;**

data\_offset **=** 0**;**

data\_byte **=** 0**;**

**end** **else** **if** **(**record\_code **==** 1**)** **begin**

// read the record type.

rec\_type**[**7**:**0**]** **=** atoh**(**file\_char**);**

**end** **else** **if** **(**record\_code **==** 2**)** **begin**

// read the upper byte of the byte count.

temp **=** atoh**(**file\_char**);**

byte\_count**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 3**)** **begin**

// read the lower byte of the byte count.

temp **=** atoh**(**file\_char**);**

byte\_count**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **>** 3**)** **begin**

**if** **(**rec\_type **==** 1**)** **begin** // If the record type is for a 16 bit address.

rec\_address**[**31**:**16**]** **=** 16'h0000**;**

**if** **(**record\_code **==** 4**)** **begin**

// read the middle byte of the address.

temp **=** atoh**(**file\_char**);**

// remove the upper most nibble since we only have single digits to represent memory addresses

rec\_address**[**15**:**12**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 5**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**11**:**8**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 6**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 7**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **begin**

// Check to see if we have reached the end of the data

**if** **(**data\_byte **<** byte\_count **-** 2 **-** 1**)** **begin** // Make sure we are less than the byte count minus the address size in bytes and checksum

// We are reading data so we want to create a lower and an upper nibble of a byte then write it to memory when we have both.

temp **=** atoh**(**file\_char**);**

rec\_data **=** rec\_data **<<** 4**;**

rec\_data**[**3**:**0**]** **=** temp**[**3**:**0**];**

nibble\_count **=** nibble\_count **+** 1**;**

**#**50**;**

**if** **(**nibble\_count **>** 1**)** **begin**

// We have both nibbles so we should write the byte to memory

// set all the lines on the falling edge of the clock.

address **=** rec\_address**+**data\_offset**;**

data\_in **=** rec\_data**;**

access\_size **=** 2'b01**;**

write **=** 1**;**

**#**100**;** // Delay one clock cycle

write **=** 0**;**

// update the data\_offest.

data\_offset **=** data\_offset **+** 1**;**

// reset the nibble count

nibble\_count **=** 0**;**

data\_byte **=** data\_byte **+** 1**;**

**end**

**end**

**end**

**end**

**if** **(**rec\_type **==** 2**)** **begin** // If the record type is for a 24 bit address.

rec\_address**[**31**:**24**]** **=** 8'h00**;**

**if** **(**record\_code **==** 4**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**23**:**20**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 5**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**19**:**16**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 6**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**15**:**12**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 7**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**11**:**8**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 8**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 9**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **begin**

// Check to see if we have reached the end of the data

**if** **(**data\_byte **<** byte\_count **-** 3 **-** 1**)** **begin** // Make sure we are less than the byte count minus the address size in bytes and checksum

// We are reading data so we want to create a lower and an upper nibble of a byte then write it to memory when we have both.

temp **=** atoh**(**file\_char**);**

rec\_data **=** rec\_data **<<** 4**;**

rec\_data**[**3**:**0**]** **=** temp**[**3**:**0**];**

nibble\_count **=** nibble\_count **+** 1**;**

**#**50**;**

**if** **(**nibble\_count **>** 1**)** **begin**

// We have both nibbles so we should write the byte to memory

// set all the lines on the falling edge of the clock.

address **=** rec\_address**+**data\_offset**;**

data\_in **=** rec\_data**;**

access\_size **=** 2'b00**;**

write **=** 1**;**

**#**100**;** // Delay one clock cycle

write **=** 0**;**

// update the data\_offest.

data\_offset **=** data\_offset **+** 1**;**

// reset the nibble count

nibble\_count **=** 0**;**

data\_byte **=** data\_byte **+** 1**;**

**end**

**end**

**end**

**end**

**if** **(**rec\_type **==** 3**)** **begin** // If the record type is for a 32 bit address.

**if** **(**record\_code **==** 4**)** **begin**

// read the upper most byte of the address.

temp **=** atoh**(**file\_char**);**

// remove the upper most nibble since we only have single digits to represent memory addresses

rec\_address**[**31**:**28**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 5**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**27**:**24**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 6**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**23**:**20**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 7**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**19**:**16**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 8**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**15**:**12**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 9**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**11**:**8**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 10**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**7**:**4**]** **=** temp**[**3**:**0**];**

**end** **else** **if** **(**record\_code **==** 11**)** **begin**

temp **=** atoh**(**file\_char**);**

rec\_address**[**3**:**0**]** **=** temp**[**3**:**0**];**

**end** **else** **begin**

// Check to see if we have reached the end of the data

**if** **(**data\_byte **<** byte\_count **-** 4 **-** 1**)** **begin** // Make sure we are less than the byte count minus the address size in bytes and checksum

// We are reading data so we want to create a lower and an upper nibble of a byte then write it to memory when we have both.

temp **=** atoh**(**file\_char**);**

rec\_data **=** rec\_data **<<** 4**;**

rec\_data**[**3**:**0**]** **=** temp**[**3**:**0**];**

nibble\_count **=** nibble\_count **+** 1**;**

**#**50**;**

**if** **(**nibble\_count **>** 1**)** **begin**

// We have both nibbles so we should write the byte to memory

// set all the lines on the falling edge of the clock.

address **=** rec\_address**+**data\_offset**;**

data\_in **=** rec\_data**;**

access\_size **=** 2'b00**;**

write **=** 1**;**

**#**100**;** // Delay one clock cycle

write **=** 0**;**

// update the data\_offest.

data\_offset **=** data\_offset **+** 1**;**

// reset the nibble count

nibble\_count **=** 0**;**

data\_byte **=** data\_byte **+** 1**;**

**end**

**end**

**end**

**end**

**end**

**#**50**;** // delay 1/2 clock cycle

// increment record\_code

record\_code **=** record\_code **+** 1**;**

// read the next character from the file.

file\_char **=** $fgetc**(**fh**);**

**end**

**end**

// Close up the file

$fclose**(**fh**);**

$monitor**(**"Done parsing the SREC file!"**);**

**end**

**always** **begin**

**#**50 clk **=** **!**clk**;**

**end**

// A function to convert ASCII upper case letters and digits to their hexadecimal value.

**function** **[**7**:**0**]**atoh**;**

**input** **[**7**:**0**]**aCode**;**

**begin**

**if** **(**aCode **>=** 8'h30 **&&** aCode **<=** 8'h39**)** **begin**

atoh **=** aCode **-** 8'h30**;**

**end** **else** **if** **(**aCode **>=** 8'h41 **&&** aCode **<=** 8'h5A**)** **begin**

atoh **=** aCode **-** 8'h37**;**

**end**

**end**

**endfunction**

**endmodule**

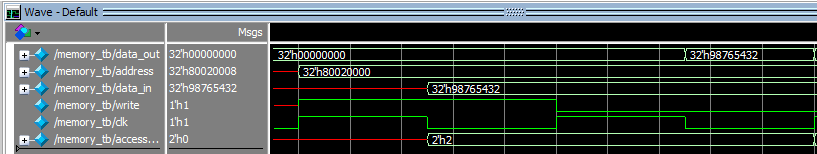
# Verification Demonstration

We split this section into two parts, the first shows that the memory module is working correctly by running our memory test bench (memory\_tb.v). Then we show that the parser is working correctly by writing the correct data to the correct addresses outlined in the SREC file.

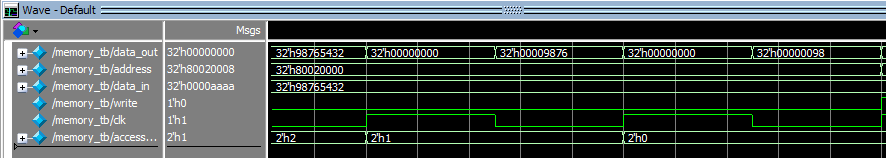
## Memory Test Bench Verification

In this verification we show that we can write an address in a single clock cycle and that we can read the value on the next clock cycle.

The first screen shot shows when we write a 32-bit value to the address 0x80020000 and we successfully read the 32-bit address on the next clock cycle.



We also show that we can read the 16 bit value and the 8 bit value of the same address in the next screenshot.



## SREC Parser Verification

We have the first 6 lines of the SREC file below for BubbleSort.c:

S0120000427562626C65536F72742E737265631E

S3158002000027BDFFC8AFBF0034AFBE003003A0F021CA

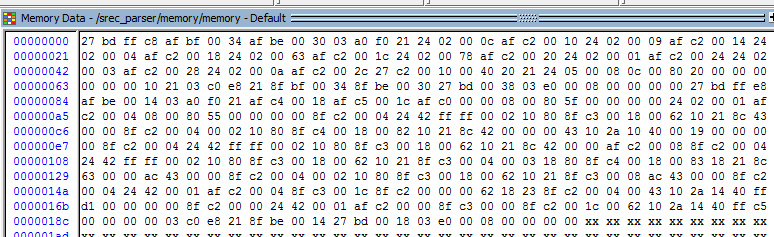
S315800200102402000CAFC2001024020009AFC20014F1

S3158002002024020004AFC2001824020063AFC2001C7F

S3158002003024020078AFC2002024020001AFC200244D

S3158002004024020003AFC200282402000AAFC2002C99

We have highlighted the data of each address in different colors and then we show a screenshot from the memory viewer with the corresponding colored boxes around the data. We also show the address portion of the SREC record in purple to help visualize what the address is. Note that in our memory viewer address 0x00000000 is equivalent to the address 0x80020000 in the SREC file since we use 0x80020000 as the offset in our memory implementation.



As we can see, each box contains 16 bytes which corresponds correctly to the data field given in the SREC file.