Final Project

Meng Wai Chan

Professor Gertner

CSc 343

May 13, 2023

Meng Wai Chan
May 13, 2023

Tables of Content

Meng Wai Chan
May 13, 2023

I. Objective

The objective of this final test project is to evaluate our ability to use LPM modules generated from Quartus to design and simulate digital computational circuits. The tasks consist of creating a Static Random-Access Memory (SRAM) using LPM with 16 words and 32 bits. Create a Add/Sub digital circuit using LPM to handle 32 bit addition and subtraction using data from a Memory Initialization File (MIF) to execute and test if our circuit is functioning.

II. Description of Specifications and Functionality

LPM SRAM of 16 words 32 bits

```
37  LIBRARY ieee;
38  USE ieee.std_logic_1164.all;
39  LIBRARY altera_mf;
40  USE altera_mf.altera_mf_components.all;
41
42  ENTITY sram IS
43     PORT
44     (
45        address     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
46        clock    : IN STD_LOGIC  := '1';
47        data     : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
48        wren     : IN STD_LOGIC ;
49        q     : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
50     );
51  END sram;
52
53  ARCHITECTURE SYN OF sram IS
54     SIGNAL sub_wire0  : STD_LOGIC_VECTOR (31 DOWNTO 0);
55
56  BEGIN
57     q    <= sub_wire0(31 DOWNTO 0);
58
59     altsyncram_component : altsyncram
60     GENERIC MAP (
61        clock_enable_input_a => "BYPASS",
62        clock_enable_output_a => "BYPASS",
63        init_file => "ram_content.mif",
64        intended_device_family => "Cyclone V",
65        lpm_hint => "ENABLE_RUNTIME_MOD=NO",
66        lpm_type => "altsyncram",
67        numwords_a => 16,
68        operation_mode => "SINGLE_PORT",
69        outdata_aclr_a => "NONE",
70        outdata_reg_a => "CLOCK0",
71        power_up_uninitialized => "FALSE",
72        read_during_write_mode_port_a => "NEW_DATA_NO_NBE_READ",
73        widthad_a => 4,
74        width_a => 32,
75        width_byteena_a => 1
76     )
77     PORT MAP (
78        address_a => address,
79        clock0 => clock,
80        data_a => data,
81        wren_a => wren,
82        q_a => sub_wire0
83     );
84  END SYN;
```

Figure 1: SRAM of 16 words 32 bits

```
DEPTH = 16; -- The size of memory in words
WIDTH = 32; -- The size of data in bits
ADDRESS_RADIX = HEX; -- The radix for address values
DATA_RADIX = BIN; -- The radix for data values
CONTENT -- start of (address : data pairs)
BEGIN

00 : 00000000000000000000000000000000; -- memory address : data
01 : 00000000000000000000000000000001;
02 : 00000000000000000000000000000010;
03 : 00000000000000000000000000000011;
04 : 00000000000000000000000000000100;
05 : 00000000000000000000000000000101;
06 : 00000000000000000000000000000110;
07 : 00000000000000000000000000000111;
08 : 00000000000000000000000000001000;
09 : 00000000000000000000000000001001;
0A : 00000000000000000000000000001010;
0B : 00000000000000000000000000001011;
0C : 00000000000000000000000000001100;
0D : 00000000000000000000000000001101;
0E : 10000000000000000000000000000000;
0F : 01111111111111111111111111111111;

END;
```

Figure 2: SRAM Memory Initialization File

LPM AddSub 32 bits

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.ALL;
3   -- Top-level entity
4   ENTITY addersubtractor2 IS
5      GENERIC (n : INTEGER := 32);
6      PORT (
7         A, B : IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
8         Clock, Reset, Sel, AddSub, Aload, Bload, Zload : IN STD_LOGIC;
9         Z, Aout, Bout: BUFFER STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
10        Overflow : OUT STD_LOGIC
11     );
12  END addersubtractor2;
13  ARCHITECTURE Behavior OF addersubtractor2 IS
14     SIGNAL G, M, Areg, Breg, Zreg : STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
15     SIGNAL SelR, AddSubR, over_flow : STD_LOGIC;
16     COMPONENT mux2to1
17        GENERIC (k : INTEGER := 32);
18        PORT (
19           V, W : IN STD_LOGIC_VECTOR(k - 1 DOWNTO 0);
20           Selm : IN STD_LOGIC;
21           F : OUT STD_LOGIC_VECTOR(k - 1 DOWNTO 0)
22        );
23     END COMPONENT;
24     COMPONENT megaddsub
25        PORT (
26           add_sub : IN STD_LOGIC;
27           dataa, datab : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
28           result : OUT STD_LOGIC_VECTOR(31 DOWNTO 0);
29           overflow : OUT STD_LOGIC
30        );
31     END COMPONENT;
```

Figure 3: LPM AddSub 32 bits Part I

```
BEGIN
   -- Define flip-flops and registers
   PROCESS (Reset, Clock)
   BEGIN
      IF Reset = '1' THEN
         Areg <= (OTHERS => '0');
         Breg <= (OTHERS => '0');
         Zreg <= (OTHERS => '0');
         SelR <= '0';
         AddSubR <= '0';
         Overflow <= '0';
      ELSIF Clock'EVENT AND Clock = '1' THEN
         SelR <= Sel;
         AddSubR <= NOT AddSub;
         Overflow <= over_flow;
         IF Aload = '1' THEN
            Areg <= A;
         END IF;
         IF Bload = '1' THEN
            Breg <= B;
         END IF;
         IF Zload = '1' THEN
            Zreg <= M;
         END IF;
      END IF;
   END PROCESS;
   -- Define combinational circuit
   nbit_addsub : megaddsub
   PORT MAP(AddSubR, G, Breg, M, over_flow);
      multiplexer :
      mux2to1
         GENERIC MAP(k => n)
   PORT MAP(Areg, Z, SelR, G);
   Z <= Zreg;
   Aout <= Areg;
   Bout <= Breg;
END Behavior;
```

Figure 4: LPM AddSub 32 bits Part II

```
-- k-bit 2-to-1 multiplexer
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY mux2to1 IS
   GENERIC (k : INTEGER := 32);
   PORT (
   V, W : IN STD_LOGIC_VECTOR(k - 1 DOWNTO 0);
   Selm : IN STD_LOGIC;
   F : OUT STD_LOGIC_VECTOR(k - 1 DOWNTO 0)
   );
END mux2to1;
ARCHITECTURE Behavior OF mux2to1 IS
BEGIN
   PROCESS (V, W, Selm)
   BEGIN
      IF Selm = '0' THEN
         F <= V;
      ELSE
         F <= W;
      END
      IF;
       END PROCESS;
    END Behavior;
```

Figure 5: LPM AddSub 32 bits Part III

```
 93   LIBRARY ieee;
 94   USE ieee.std_logic_1164.ALL;
 95   LIBRARY lpm;
 96   USE lpm.lpm_components.ALL;
 97  ENTITY megaddsub IS
 98      PORT (
 99          add_sub : IN STD_LOGIC;
100          dataa : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
101          datab : IN
102          STD_LOGIC_VECTOR (31 DOWNTO 0);
103          result : OUT
104          STD_LOGIC_VECTOR (31 DOWNTO 0);
105          overflow : OUT
106          STD_LOGIC
107          );
108  END megaddsub;
109 ARCHITECTURE SYN OF megaddsub IS
110  SIGNAL sub_wire0 : STD_LOGIC;
111  SIGNAL sub_wire1 : STD_LOGIC_VECTOR (31 DOWNTO 0);
112 COMPONENT lpm_add_sub
113     GENERIC (
114          lpm_width : NATURAL;
115          lpm_direction : STRING;
116          lpm_type : STRING;
117          lpm_hint :
118          STRING
119      );
```

Figure 6: LPM AddSub 32 bits Part IV

```
120         PORT (
121            dataa : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
122            add_sub : IN STD_LOGIC;
123            datab : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
124            overflow : OUT STD_LOGIC;
125            result : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
126     );
127 END
128 COMPONENT;
129 BEGIN
130     overflow <= sub_wire0;
131     result <= sub_wire1(31 DOWNTO 0);
132     lpm_add_sub_component :
133     lpm_add_sub
134        GENERIC MAP(
135        lpm_width => 32,
136        lpm_direction => "UNUSED",
137        lpm_type => "LPM_ADD_SUB",
138        lpm_hint => "ONE_INPUT_IS_CONSTANT=NO,CIN_USED=NO")
139        PORT MAP(
140           dataa => dataa,
141           add_sub => add_sub,
142           datab => datab,
143           overflow => sub_wire0,
144           result => sub_wire1
145        );
146  END SYN;
```

Figure 7: LPM AddSub 32 bits Part V

The 32 bits add/subtractor is able to add 32 bits number reading from given data A, and data B and perform operation based on the signal “AddSub”, addition if “AddSub” = 0, and subtraction if “AddSub” = 1, this will perform operation when signal “Zload” is called to load the new value to Z based on the “AddSub” signal. Signal “Aload” and “Bload” are used to load value to A and B from the input “SRAM” by using demux to

select which number the SRAM will input into this 32 bit Add/Substractor. The overflow signal is used to alert the user if an overflow occurs.

Demux 1 to 2

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3
4  entity demux1to2 is
5      generic (n : INTEGER := 32);
6      port ( i : in  std_logic_vector(n-1 downto 0);
7             sel   : in  std_logic;
8             o1 : out  std_logic_vector(n-1 downto 0);
9             o2 : out  std_logic_vector(n-1 downto 0));
10 end demux1to2;
11
12 architecture demux1to2_arch of demux1to2 is
13 begin
14     process (i, sel)
15     begin
16         case sel is
17             when '0' =>
18                 o1 <= i;
19                 o2 <= (others => '0');
20             when '1' =>
21                 o1 <= (others => '0');
22                 o2 <= i;
23             when others =>
24                 o1 <= (others => '0');
25                 o2 <= (others => '0');
26         end case;
27     end process;
28 end demux1to2_arch;
```

Figure 8: Demultiplexer 1 to 2

The Demux is used to connect the SRAM to the add/subtractor for reading the value from the SRAM based on a given address.

Meng Wai Chan
May 13, 2023

Test Bench

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity addersubtractor2_tb is
6   end addersubtractor2_tb;
7
8   architecture addersubtractor2_tb_arch of addersubtractor2_tb is
9
10  COMPONENT sram IS
11      PORT
12      (
13          address     : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
14          clock       : IN STD_LOGIC  := '1';
15          data        : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
16          wren        : IN STD_LOGIC ;
17          q           : OUT STD_LOGIC_VECTOR (31 DOWNTO 0)
18      );
19  END COMPONENT;
20
21  COMPONENT addersubtractor2 IS
22      GENERIC (n : INTEGER := 32);
23      PORT (
24          A, B : IN STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
25          Clock, Reset, Sel, AddSub, Aload, Bload, Zload : IN STD_LOGIC;
26          Z, Aout, Bout : BUFFER STD_LOGIC_VECTOR(n - 1 DOWNTO 0);
27          Overflow : OUT STD_LOGIC
28      );
29  END COMPONENT;
```

Figure 9: Test Bench Part I

```
31  COMPONENT demux1to2 is
32      GENERIC (n : INTEGER := 32);
33      PORT ( i : in  std_logic_vector(n-1 downto 0);
34             sel   : in  std_logic;
35             o1 : out  std_logic_vector(n-1 downto 0);
36             o2 : out  std_logic_vector(n-1 downto 0));
37  END COMPONENT;
38
39
40     signal address: std_logic_vector(3 downto 0) := "0000";
41     signal clock, reg_sel, Reset, Sel, Aload, Bload, Zload, AddSub, Overflow: std_logic :='0';
42     signal data_in, data_out, A, B, Z, Aout, Bout: std_logic_vector(31 downto 0);
```

Figure 10: Test Bench Part II

```
begin

   sram_inst: sram
   port map (
   address => address,
   clock => clock,
   data => data_in,
   wren => '0',
   q => data_out
   );

   demux_inst: demux1to2
   port map(
   i => data_out,
   sel => reg_sel,
   o1 => A,
   o2 => B
   );

   addersubtractor2_inst: addersubtractor2
   port map(
   A => A,
   B => B,
   Clock => clock,
   Reset => Reset,
   Sel => Sel,
   AddSub => AddSub,
   Aload => Aload,Bload => Bload,Zload => Zload,
   Z => Z,
   Aout => Aout,
   Bout => Bout,
   Overflow => Overflow
   );

   process
     begin
       clock <= '0';
       wait for 5 ns;
       clock <= '1';
       wait for 5 ns;
   end process;
```

Figure 11: Test Bench Part III

This Test bench will perform every operation every 5 ns. This testbench file is used to connect all the necessary components together in order to perform addition and subtraction using value within the SRAM.

Addition of Two Integers X1 + X2 VHDL

```
88      process
89      begin
90          -- Test Case 1 x1 + x2
91          -- 100 ns Total
92          Aload <= '1';
93          address <= "1010"; -- address 10
94          wait for 20 ns;
95
96          wait for 10 ns;
97          Aload <='0';
98          wait for 10 ns;
99          reg_sel <= '1';
100         address <= "0010"; -- address 2
101         wait for 20 ns;
102         Bload <= '1';
103         wait for 20 ns;
104         Bload <= '0';
105
106         Zload <='1';
107         wait for 10 ns;
108         Zload <= '0';
109         reg_sel <= '0';
110         wait for 10 ns;
111
112         -- Test Case 1 Ended
113         -- Reset for next Test case
114         Reset <= '1';
115         wait for 10 ns;
116         Reset <= '0';
117         wait for 10 ns;
```

Figure 12: Add X1 + X2

Meng Wai Chan
May 13, 2023

Subtraction of Two Integers X1 - X2 VHDL

```
119         -- Test Case 2 x1 - x2
120         -- 110 ns
121         AddSub <= '1';
122         wait for 10 ns;
123         Aload <= '1';
124         address <= "1100"; -- address 12
125         wait for 20 ns;
126
127         wait for 10 ns;
128         Aload <='0';
129         wait for 10 ns;
130         reg_sel <= '1';
131         address <= "0110"; -- address 6
132         wait for 20 ns;
133         Bload <= '1';
134         wait for 20 ns;
135         Bload <= '0';
136
137         Zload <='1';
138         wait for 10 ns;
139         Zload <= '0';
140         reg_sel <= '0';
141         AddSub <= '0';
142         wait for 10 ns;
143
144         -- Test Case 2 Ended
145         -- Reset for next Test Case
146         Reset <= '1';
147         wait for 10 ns;
148         Reset <= '0';
149         wait for 10 ns;
```

Figure 13: Sub X1 - X2

Meng Wai Chan
May 13, 2023

Cumulative Sum X1 + X2 + X3 + X4 VHDL

```
151         -- Test Case 3 x1 + x2 + x3 + x4
152         -- 200 ns
153         Aload <= '1';
154         address <= "0001"; -- address 1
155         wait for 20 ns;
156
157         wait for 10 ns;
158         Aload <= '0';
159         wait for 10 ns;
160         reg_sel <= '1';
161         address <= "0010"; -- address 2
162         wait for 20 ns;
163         Bload <= '1';
164         wait for 20 ns;
165         Bload <= '0';
166         zload <= '1';
167         Sel <= '1';
168         wait for 10 ns;
169
170         zload <= '0';
171         address <= "0011"; -- Load data from address 3
172         wait for 20 ns;
173         Bload <= '1';
174         wait for 20 ns;
175         Bload <= '0';
176         Zload <= '1';
177         wait for 10 ns;
178         zload <='0';
179         address <= "0100"; -- Load data from address 4
180         wait for 20 ns;
181         Bload <= '1';
182         wait for 20 ns;
183         Bload <= '0';
184         zload <='1';
185         wait for 10 ns;
186         zload <= '0';
187         reg_sel <= '0';
188         Sel <= '0';
189         wait for 10 ns;
190
191         -- Test Case 3 Ended
192         -- Reset for next Test Case
193         Reset <= '1';
194         wait for 10 ns;
195         Reset <= '0';
196         wait for 10 ns;
```

Figure 14: Add X1 + X2 + X3 + X4

Meng Wai Chan
May 13, 2023

Cumulative Subtract X1 - X2 - X3 - X4 VHDL

```
198         -- Test Case 4 x1 - x2 - x3 - x4
199         -- 210 ns
200         AddSub <= '1';
201         wait for 10 ns;
202         Aload <= '1';
203         address <= "1101"; -- Load data from address 13
204         wait for 20 ns;
205
206         wait for 10 ns;
207         Aload <= '0';
208         wait for 10 ns;
209         reg_sel <= '1';
210         address <= "0011"; -- Load data from address 3
211         wait for 20 ns;
212         Bload <= '1';
213         wait for 20 ns;
214         Bload <= '0';
215         zload <= '1';
216         Sel <= '1';
217         wait for 10 ns;
218         zload <= '0';
219         address <= "0010"; -- Load data from address 2
220         wait for 20 ns;
221         Bload <= '1';
222         wait for 20 ns;
223         Bload <= '0';
224         Zload <= '1';
225         wait for 10 ns;
226         zload <='0';
227         address <= "0001"; -- Load data from address 1
228         wait for 20 ns;
229         Bload <= '1';
230         wait for 20 ns;
231         Bload <= '0';
232         zload <='1';
233         wait for 10 ns;
234         zload <= '0';
235         reg_sel <= '0';
236         AddSub <= '0';
237         Sel <= '0';
238         wait for 10 ns;
239
240         -- Reset for next Test Case
241         Reset <= '1';
242         wait for 10 ns;
243         Reset <= '0';
244         wait for 10 ns;
```

Figure 15: Sub X1 - X2 - X3 - X4

Overflow VHDL

```
246         -- Test Case 5 Overflow
247         Aload <= '1';
248         address <= "1111"; -- address 16
249         wait for 20 ns;
250 
251         wait for 10 ns;
252         Aload <='0';
253         wait for 10 ns;
254         reg_sel <= '1';
255         address <= "0001"; -- address 1
256         wait for 20 ns;
257         Bload <= '1';
258         wait for 20 ns;
259         Bload <= '0';
260 
261         Zload <='1';
262         wait for 10 ns;
263         Zload <= '0';
264         reg_sel <= '0';
265         wait for 10 ns;
266 
267     end process;
```

Figure 16: Overflow

III. Simulation

Addition of Two Integers X1 + X2

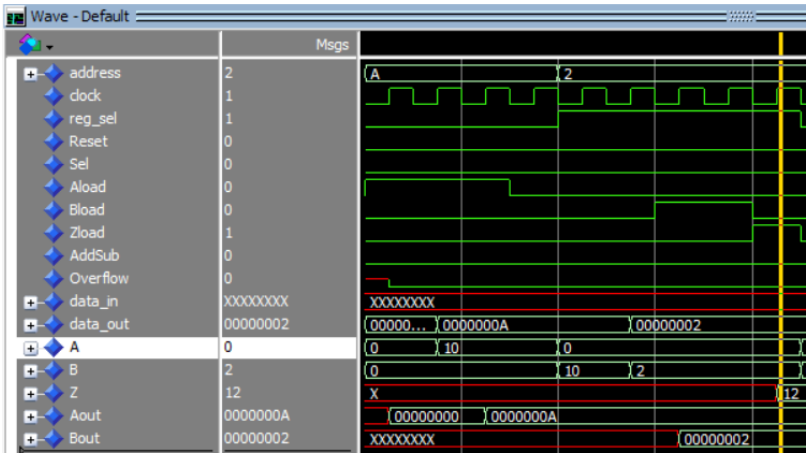

Figure 17: Add X1 + X2 Simulation

As you can see we first load from address 0000000A into A which is the value 10 in decimal, after we which turn the signal “reg_sel” to load the second value from address

00000002 into B. then we perform the adding operation by loading their sum into Z.

which is 10 + 2 = 12.

Subtraction of Two Integers X1 - X2

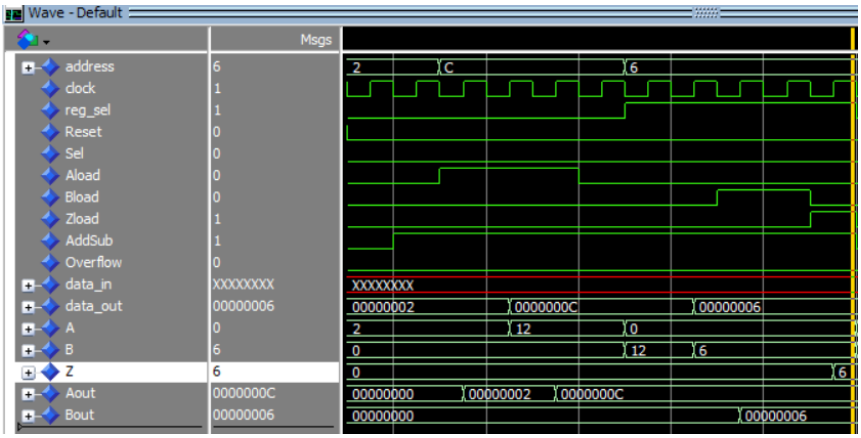

Figure 18: Sub X1 - X2 Simulation

This simulation is similar to the Addition as we load the number to A and B, then perform the operation by calling the “Zload” signal, to output the final value into Z.

Cumulative Sum X1 + X2 + X3 + X4

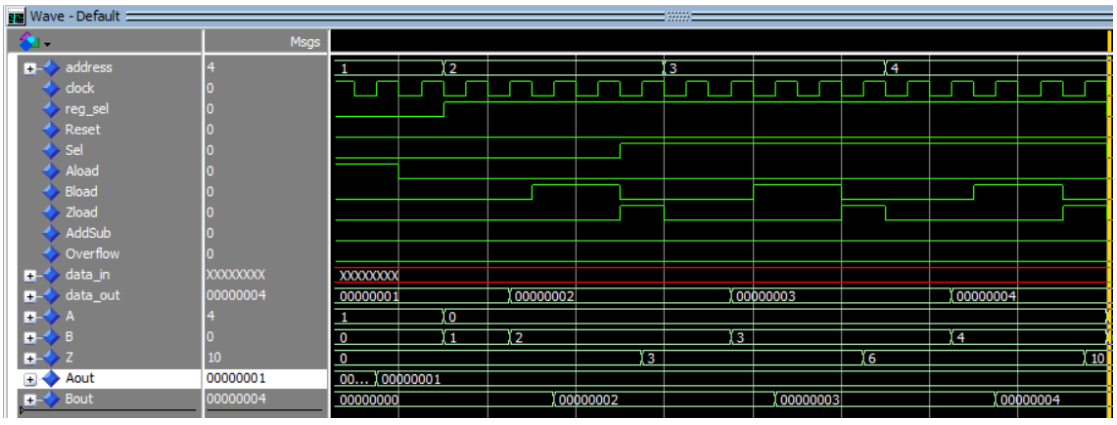

Figure 19: Add X1 + X2 + X3 + X4 Simulation

This simulation is performing 1 + 2 + 3 + 4. Which respectively are 00000001, 00000002, 00000003. 00000004, addresses from the MIF file, the result 10 is outputted into Z.

Cumulative Subtract X1 - X2 - X3 - X4

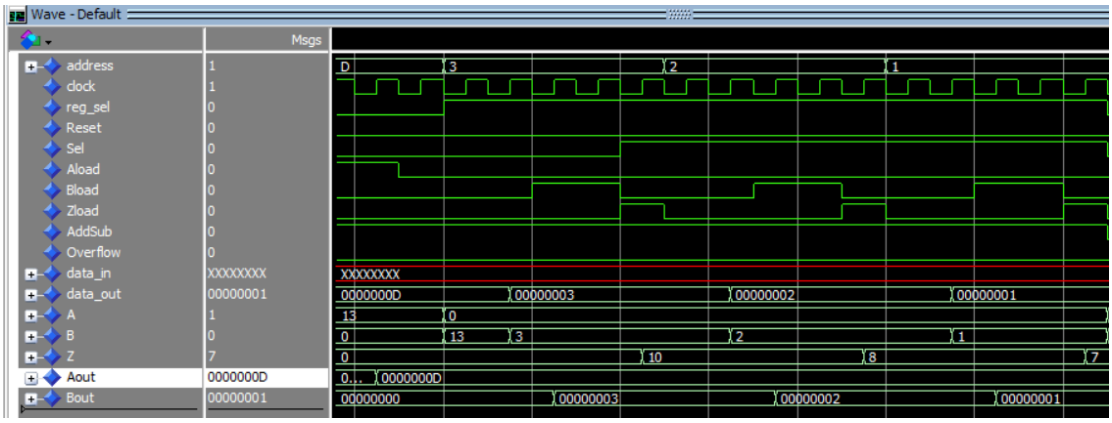

Figure 20: Sub X1 - X2 - X3 - X4 Simulation

This simulation is performing 13 - 3 - 2 - 1. Which respectively are 0000000C, 00000003, 00000002, 00000001, addresses from the MIF file, the result 10 is outputted into Z.

Overflow

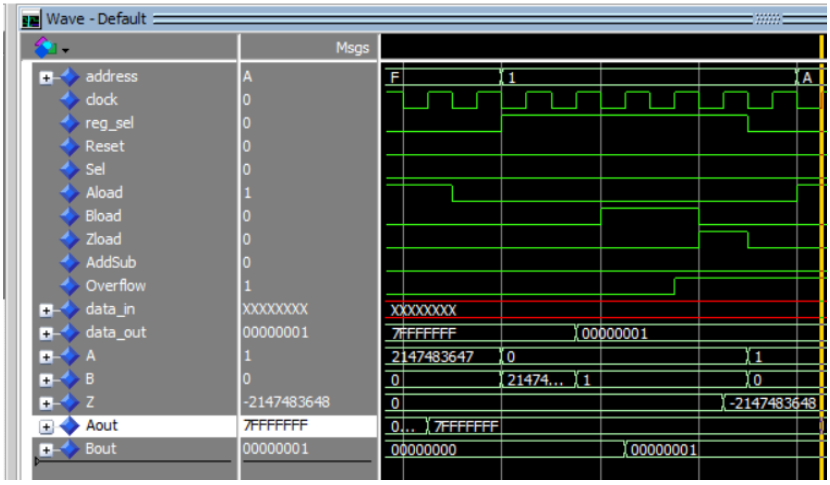

Figure 21: Overflow Simulation

The overflow demonstration is adding the largest possible integer + 1, which is 7FFFFFFF + 00000001 which is 80000000, and the overflow signal is turned into 1.

IV. Conclusion

In this lab, we used SRAM as a memory to store data at a specified address. The SRAM is connected to a demultiplexer which directs the data to either input A or input B of an adder-subtractor based on the value of the **'reg_sel'** signal. The adder-subtractor uses a MegaAddSub component and a multiplexer to allow for selecting either the output Z or input A values using the **'Sel'** signal. By setting the **'Sel'** signal, we can select Z, which allows for cumulative addition and subtraction. We tested the circuit using basic arithmetic operations such as addition and subtraction of two numbers, as well as cumulative addition and subtraction. Additionally, we tested the **'Overflow'** signal by adding a number to the most positive number, which triggers the **'Overflow'** signal to indicate that the result is out of range and cannot be represented using the given number of bits.