Lab 07.

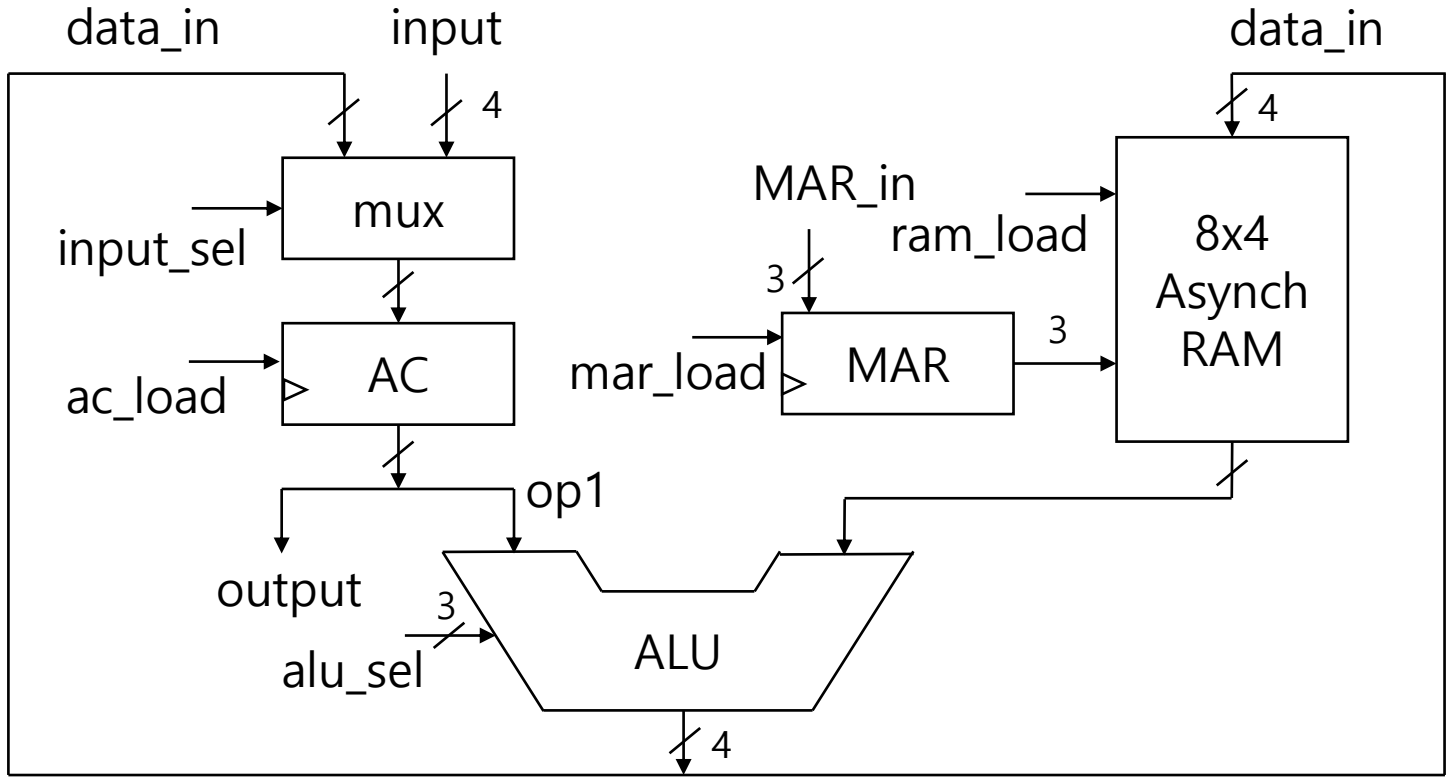
RAM 기반 Data Processor 구현

201810800 이혜인

Block diagram

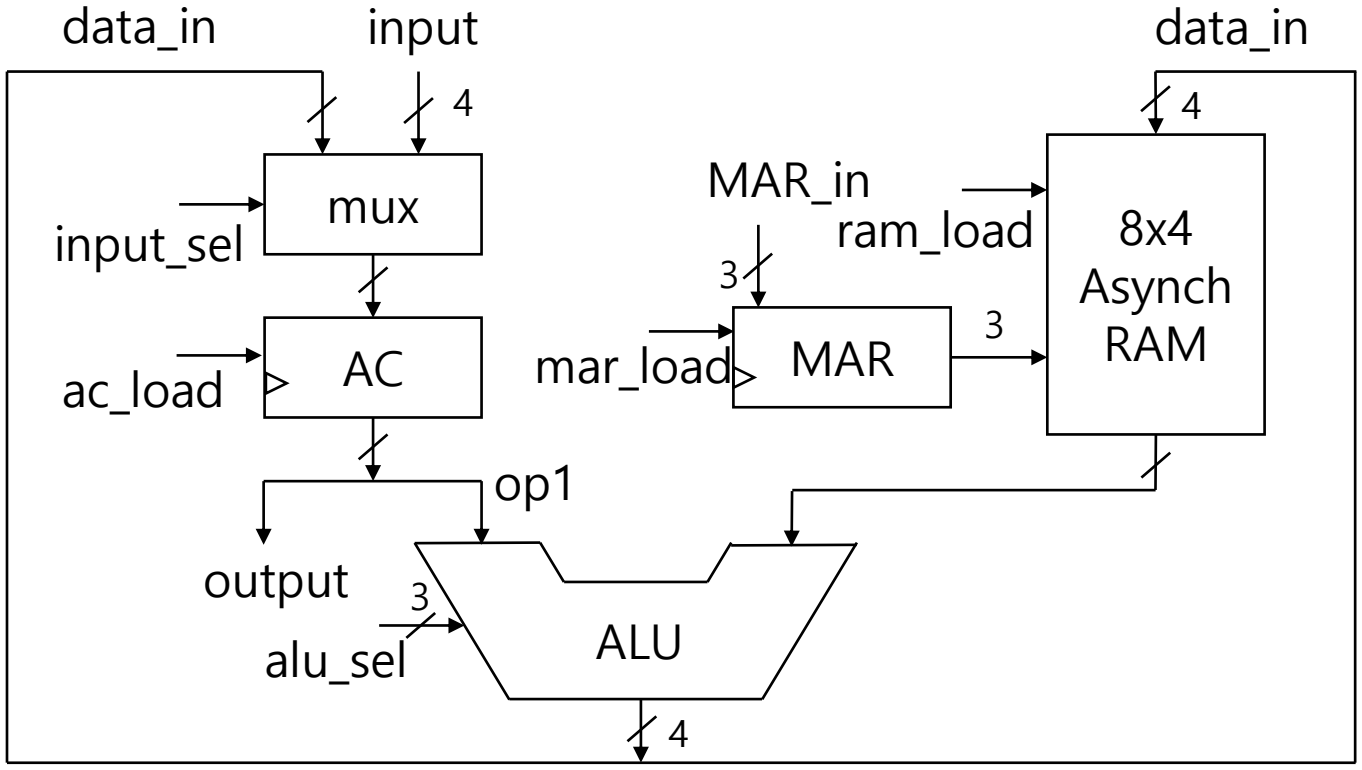
- 4 bit data
- AC (accumulator) & MAR (memory address register) registers
- 8x4 Asynch RAM
- 3 bit alu_sel

- 다음의 동작을 순차적으로 수행하도록 simulation 하고 검증하라.
  - ac ← input ('3')
  - M[2] ← ac
  - ac ← input ('2')
  - M[1] ← ac
  - ac ← input ('4')
  - ac ← ac + M[1]
  - ac ← ac + M[2]



실습 지침

- ALU는 주어진 code (simple_alu.vhd)를 활용
- RAM은 지난 실습자료 code (asynch_ram.vhd)활용
- mux와 AC, MAR는 간단함. 직접 구현.
- Top-level design entity는 structural description을 이용하여 구현.
- Simulation에 시간이 많이 소요. Test pattern은 transcript를 copy-재활용하여 반복 입력.
- 각 동작(instruction)을 수행하는데 필요한 control input들 (input_sel, ac_load, mar_load, alu_sel, ram_load 등)과 외부 data 입력 ("input"과 MAR_in)을 명시하라.
- 필요하면 내부 signal들을 외부로 뽑아서 관찰할 수 있게 하라.



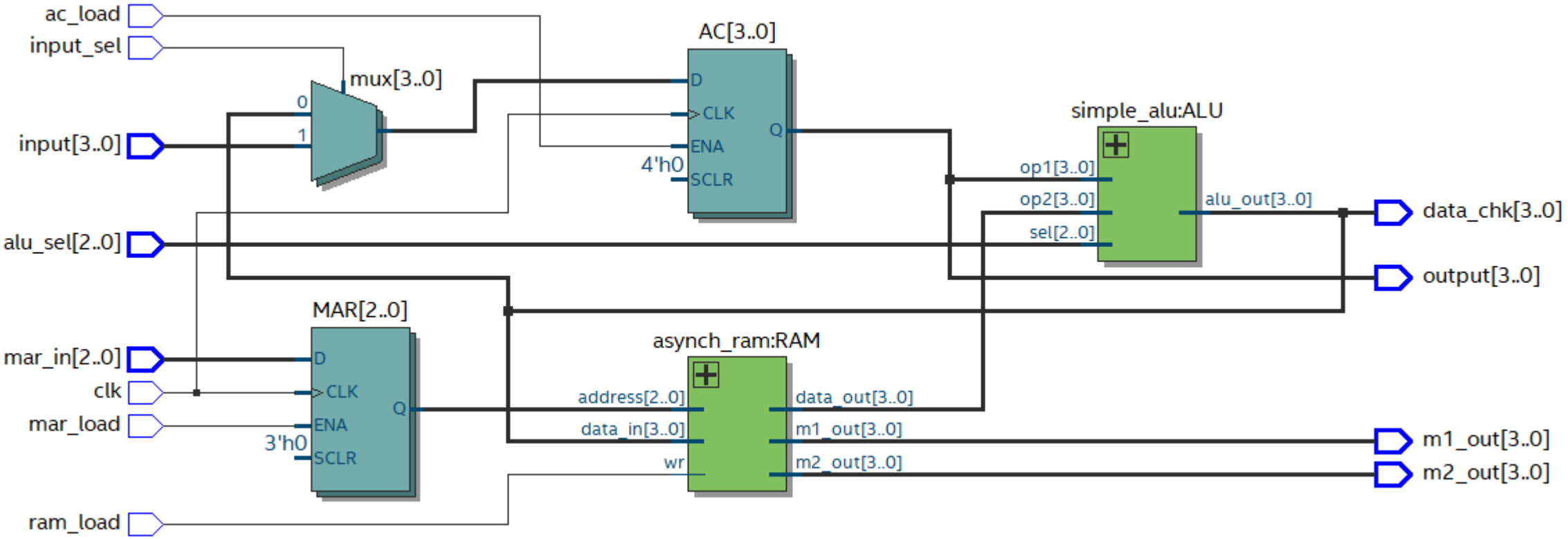
Structural description example

```
ARCHITECTURE sample OF data_processor1 IS
…
        component asynch_ram
                port (
                        data_in: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
                        address: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
                        wr: IN STD_LOGIC;
                        data_out: OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
        end component;

…
BEGIN
…

        RAM : asynch_ram
                port map (data_in => data_in, address => MAR, wr => ram_load, data_out =>
data_out );
```

RTL view example



1. VHDL Code – Asynchronous RAM

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  ENTITY asynch_ram IS
6   PORT (  data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
7           address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8           wr       : IN STD_LOGIC;
9           data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
10          m1_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
11          m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
12 END asynch_ram;
13
14 ARCHITECTURE rtl OF asynch_ram IS
15  TYPE MEM IS ARRAY(0 TO 7) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
16  SIGNAL ram_block: MEM;
17 BEGIN
18  PROCESS (wr, data_in, address)
19  BEGIN
20    IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
21       data_out <= (others => 'Z');
22    else
23       data_out <= ram_block(to_integer(unsigned(address)));
24   END IF;
25  END PROCESS;
26  m1_out <= ram_block(2);
27  m2_out <= ram_block(1);
28 END rtl;
```

→ 다음은 강의자료에 주어진 Asynchronous RAM Code를 8X4 RAM으로 바꿔서 나타낸 VHDL Code이다.

→ 8X4 RAM이므로 MEM(memory)의 ARRAY 개수는 0 TO 7로 8개, bit 수는 3 DOWNTO 0로 4 bit word를 가진다.

→ 4 bit word를 8개 쌓은 ARRAY

1. VHDL Code – Asynchronous RAM

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY asynch_ram IS
6    PORT (  data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
7            address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8            wr       : IN STD_LOGIC;
9            data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
10           m1_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
11           m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
12  END asynch_ram;
13
14  ARCHITECTURE rtl OF asynch_ram IS
15   TYPE MEM IS ARRAY(0 TO 7) OF STD_LOGIC_VECTOR(3 DOWNTO 0);
16   SIGNAL ram_block: MEM;
17  BEGIN
18   PROCESS (wr, data_in, address)
19   BEGIN
20     IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
21        data_out <= (others => 'Z');
22     else
23        data_out <= ram_block(to_integer(unsigned(address)));
24    END IF;
25   END PROCESS;
26   m1_out <= ram_block(2);
27   m2_out <= ram_block(1);
28  END rtl;
```

→ 추가적으로 M[1]과 M[2]에 있는 data 값이 제대로 들어갔는지 확인하기 위해 m1_out과 m2_out을 생성하였다.

→ m1_out은 M[2]를 확인 할 수 있도록 ram_block(2)를 넣고, m1_out은 M[1]을 확인할 수 있도록 ram_block(1)을 넣었다.

1. VHDL Code - ALU

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4
5   ENTITY simple_alu IS
6    PORT (  op1, op2 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
7            sel      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
8            alu_out  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
9   END simple_alu;
10
11  ARCHITECTURE sample OF simple_alu IS
12  BEGIN
13   PROCESS (sel, op1, op2)
14   BEGIN
15     case sel is
16        when "000" => alu_out <= op1;
17        when "001" => alu_out <= op2;
18        when "010" => alu_out <= op1 + op2;
19        when "011" => alu_out <= op1 - op2;
20        when "100" => alu_out <= op1 and op2;
21        when "101" => alu_out <= op1 or op2;
22        when "110" => alu_out <= op1 xor op2;
23        when others => alu_out <= not op1;
24    end case;
25   END PROCESS;
26  END sample;
```

→ 다음은 강의자료에 주어진 Simple ALU Code이다.

→ 해당 Code는 다음과 같은 'sel'값을 통해 어떤 operation(연산)을 사용하여 계산할 것인지 정하는 Code이다.

1. VHDL Code – Data Processor

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   ENTITY lab_07 IS
7    PORT (  input_sel    : IN STD_LOGIC ;
8            ac_load     : IN STD_LOGIC ;
9            mar_load    : IN STD_LOGIC ;
10           alu_sel     : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
11           ram_load    : IN STD_LOGIC ;
12           input       : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
13           MAR_in      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
14           clk         : IN STD_LOGIC ;
15           data_check  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
16           output      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
17           ml_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
18           m2_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
19           mux_out     : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
20           mar_out     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
21  END lab_07;
22
23  ARCHITECTURE data_processor OF lab_07 IS
24     SIGNAL MAR : STD_LOGIC_VECTOR (2 DOWNTO 0);
25     SIGNAL ram_out : STD_LOGIC_VECTOR(3 DOWNTO 0);
26     SIGNAL AC : STD_LOGIC_VECTOR(3 DOWNTO 0);
27     SIGNAL alu_out_signal : STD_LOGIC_VECTOR(3 DOWNTO 0);
28     SIGNAL mux : STD_LOGIC_VECTOR(3 DOWNTO 0);
29
30     component asynch_ram
31        port (   data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
32                 address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
33                 wr       : IN STD_LOGIC;
34                 data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
35                 ml_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
36                 m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
37     end component;
```

```
39     component simple_alu
40        PORT (   op1, op2 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
41                 sel      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
42                 alu_out  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
43     end component;
44
45  BEGIN
46     RAM : asynch_ram
47        port map (data_in => alu_out_signal, address => MAR, wr => ram_load, data_out => ram_out, ml_out=>ml_out, m2_out=>m2_out);
48     ALU : simple_alu
49        port map (op1 => AC, op2 => ram_out, sel => alu_sel, alu_out => alu_out_signal) ;
50  PROCESS (mar_load, ac_load, input_sel, clk)
51  BEGIN
52
53     --AC
54     IF clk'EVENT AND clk = '1' THEN
55        IF ac_load = '1' THEN
56           AC <= mux;
57        END IF;
58     END IF;
59
60     --MAR
61     IF clk'EVENT AND clk = '1' THEN
62        IF mar_load = '1' THEN
63           MAR <= Mar_in;
64        END IF;
65     END IF;
67       --MUX
68     IF input_sel = '1' THEN
69        mux <= input;
70     ELSE
71        mux <= alu_out_signal;
72     END IF;
73  END PROCESS;
74  output <= AC;
75  data_check <= alu_out_signal;
76  mux_out <= mux;
77  mar_out <= MAR;
78 END data_processor;
```

→ 다음은 위의 코드를 component 하고 각각 AC, MAR, MUX의 기능을 구현한 VHDL Code이다.

→ 이는 **Structural description example**을 참고하여 구현하였다.

1. VHDL Code – Data Processor

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   ENTITY lab_07 IS
7    PORT (  input_sel    : IN STD_LOGIC ;
8            ac_load     : IN STD_LOGIC ;
9            mar_load    : IN STD_LOGIC ;
10           alu_sel     : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
11           ram_load    : IN STD_LOGIC ;
12           input       : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
13           MAR_in      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
14           clk         : IN STD_LOGIC ;
15           data_check  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
16           output      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
17           ml_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
18           m2_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
19           mux_out     : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
20           mar_out     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
21  END lab_07;
22
23  ARCHITECTURE data_processor OF lab_07 IS
24     SIGNAL MAR : STD_LOGIC_VECTOR (2 DOWNTO 0);
25     SIGNAL ram_out : STD_LOGIC_VECTOR(3 DOWNTO 0);
26     SIGNAL AC : STD_LOGIC_VECTOR(3 DOWNTO 0);
27     SIGNAL alu_out_signal : STD_LOGIC_VECTOR(3 DOWNTO 0);
28     SIGNAL mux : STD_LOGIC_VECTOR(3 DOWNTO 0);
29
30     component asynch_ram
31        port (   data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
32                 address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
33                 wr       : IN STD_LOGIC;
34                 data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
35                 ml_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
36                 m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
37     end component;
```

→ 해당 Input과 Output을 나열해 놓은 코드이다.

→ 해당 Input으로는 input_sel, ac_load, mar_load, alu_sel, ram_load, input, MAR_in, clk를 가지고 있고, 각각 Clock과 직접 input값을 넣어주는 Input, 그리고 input 값을 반영하는 여부를 결정해주는 Input이다.

1. VHDL Code – Data Processor

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   ENTITY lab_07 IS
7    PORT (  input_sel    : IN STD_LOGIC ;
8            ac_load     : IN STD_LOGIC ;
9            mar_load    : IN STD_LOGIC ;
10           alu_sel     : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
11           ram_load    : IN STD_LOGIC ;
12           input       : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
13           MAR_in      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
14           clk         : IN STD_LOGIC ;
15           data_check  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
16           output      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
17           m1_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
18           m2_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
19           mux_out     : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
20           mar_out     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
21  END lab_07;
22
23  ARCHITECTURE data_processor OF lab_07 IS
24     SIGNAL MAR : STD_LOGIC_VECTOR (2 DOWNTO 0);
25     SIGNAL ram_out : STD_LOGIC_VECTOR(3 DOWNTO 0);
26     SIGNAL AC : STD_LOGIC_VECTOR(3 DOWNTO 0);
27     SIGNAL alu_out_signal : STD_LOGIC_VECTOR(3 DOWNTO 0);
28     SIGNAL mux : STD_LOGIC_VECTOR(3 DOWNTO 0);
29
30     component asynch_ram
31        port (   data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
32                 address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
33                 wr       : IN STD_LOGIC;
34                 data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
35                 m1_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
36                 m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
37     end component;
```

→ 해당 Output으로는 data_check, output, m1_out, m2_out, mux_out, mar_out이 존재하는데 이는 내부 Signal의 값이 적절한 값을 가지는지 판단하기 위해 작성한 Output이다.

1. VHDL Code – Data Processor

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   ENTITY lab_07 IS
7    PORT (  input_sel    : IN STD_LOGIC ;
8            ac_load     : IN STD_LOGIC ;
9            mar_load    : IN STD_LOGIC ;
10           alu_sel     : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
11           ram_load    : IN STD_LOGIC ;
12           input       : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
13           MAR_in      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
14           clk         : IN STD_LOGIC ;
15           data_check  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
16           output      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
17           ml_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
18           m2_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
19           mux_out     : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
20           mar_out     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
21  END lab_07;
22
23  ARCHITECTURE data_processor OF lab_07 IS
24    SIGNAL MAR : STD_LOGIC_VECTOR (2 DOWNTO 0);
25    SIGNAL ram_out : STD_LOGIC_VECTOR(3 DOWNTO 0);
26    SIGNAL AC : STD_LOGIC_VECTOR(3 DOWNTO 0);
27    SIGNAL alu_out_signal : STD_LOGIC_VECTOR(3 DOWNTO 0);
28    SIGNAL mux : STD_LOGIC_VECTOR(3 DOWNTO 0);
29
30    component asynch_ram
31       port (   data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
32                address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
33                wr       : IN STD_LOGIC;
34                data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
35                ml_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
36                m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
37    end component;
```

→ Architecture 내부에 존재하는 Signal들은 내부와 외부의 Signal들을 용도에 맞게 연결해주기 위한 내부 Signal이다.

→ MAR : MAR의 output을 RAM의 address와 연결해주는 Signal

→ ram_out : RAM의 해당 address에 있는 data output을 ALU와 연결해주는 Signal

→ AC : AC의 output을 ALU의 op1과 연결해주는 Signal

1. VHDL Code – Data Processor

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.std_logic_unsigned.all;
4   use ieee.numeric_std.all;
5
6   ENTITY lab_07 IS
7    PORT (  input_sel    : IN STD_LOGIC ;
8            ac_load     : IN STD_LOGIC ;
9            mar_load    : IN STD_LOGIC ;
10           alu_sel     : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
11           ram_load    : IN STD_LOGIC ;
12           input       : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
13           MAR_in      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
14           clk         : IN STD_LOGIC ;
15           data_check  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
16           output      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
17           m1_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
18           m2_out      : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
19           mux_out     : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
20           mar_out     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0));
21  END lab_07;
22
23  ARCHITECTURE data_processor OF lab_07 IS
24    SIGNAL MAR : STD_LOGIC_VECTOR (2 DOWNTO 0);
25    SIGNAL ram_out : STD_LOGIC_VECTOR(3 DOWNTO 0);
26    SIGNAL AC : STD_LOGIC_VECTOR(3 DOWNTO 0);
27    SIGNAL alu_out_signal : STD_LOGIC_VECTOR(3 DOWNTO 0);
28    SIGNAL mux : STD_LOGIC_VECTOR(3 DOWNTO 0);
29
30    component asynch_ram
31      port (   data_in  : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
32               address  : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
33               wr       : IN STD_LOGIC;
34               data_out : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
35               m1_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
36               m2_out   : OUT STD_LOGIC_VECTOR (3 DOWNTO 0));
37    end component;
```

→ alu_out_signal : ALU의 output을 각각 RAM과 MUX와 연결해주는 Signal

→ mux : mux의 output 값을 AC의 Input과 연결해주는 Signal

→ asynch_ram을 component를 하였다.

(asynch_ram에 있는 port를 가져와서 그대로 작성하였다.)

1. VHDL Code – Data Processor

```
39  component simple_alu
40     PORT (   op1, op2 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
41              sel      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
42              alu_out  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
43  end component;
44
45  BEGIN
46     RAM : asynch_ram
47        port map (data_in => alu_out_signal, address => MAR, wr => ram_load, data_out => ram_out, m1_out=>m1_out, m2_out=>m2_out);
48     ALU : simple_alu
49        port map (op1 => AC, op2 => ram_out, sel => alu_sel, alu_out => alu_out_signal) ;
50  PROCESS (mar_load, ac_load, input_sel, clk)
51  BEGIN
52
53     --AC
54     IF clk'EVENT AND clk = '1' THEN
55        IF ac_load = '1' THEN
56           AC <= mux;
57        END IF;
58     END IF;
59
60     --MAR
61     IF clk'EVENT AND clk = '1' THEN
62        IF mar_load = '1' THEN
63           MAR <= Mar_in;
64        END IF;
65     END IF;
67       --MUX
68     IF input_sel = '1' THEN
69        mux <= input;
70     ELSE
71        mux <= alu_out_signal;
72     END IF;
73  END PROCESS;
74  output <= AC;
75  data_check <= alu_out_signal;
76  mux_out <= mux;
77  mar_out <= MAR;
78 END data_processor;
```

ı를 component하였다.

m에 있는 port를 가져와서 그대로

-.)

→ 각각의 RAM과 ALU Signal들을 위에서 기재한 내부 Signal들과 연결을 시켜주었다.

1. VHDL Code – Data Processor

```
39    component simple_alu
40       PORT (   op1, op2 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
41                sel      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
42                alu_out  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
43    end component;
44
45  BEGIN
46     RAM : asynch_ram
47        port map (data_in => alu_out_signal, address => MAR, wr => ram_load, data_out => ram_out, m1_out=>m1_out, m2_out=>m2_out);
48     ALU : simple_alu
49        port map (op1 => AC, op2 => ram_out, sel => alu_sel, alu_out => alu_out_signal) ;
50  PROCESS (mar_load, ac_load, input_sel, clk)
51  BEGIN
52
53    --AC
54    IF clk'EVENT AND clk = '1' THEN
55       IF ac_load = '1' THEN
56          AC <= mux;
57       END IF;
58    END IF;
59
60    --MAR
61    IF clk'EVENT AND clk = '1' THEN
62       IF mar_load = '1' THEN
63          MAR <= Mar_in;
64       END IF;
65    END IF;
67      --MUX
68    IF input_sel = '1' THEN
69       mux <= input;
70    ELSE
71       mux <= alu_out_signal;
72    END IF;
73  END PROCESS;
74  output <= AC;
75  data_check <= alu_out_signal;
76  mux_out <= mux;
77  mar_out <= MAR;
78  END data_processor;
```

① RAM

→ data_in은 ALU의 Output값인 alu_out_signal과 연결하고, address는 MAR의 output값인 MAR과 연결한다.

→ Ram의 write 여부를 결정하는 wr은 ram_load와 연결하고, data_out은 ram_out과 연결한다.

→ m1_out과 m2_out은 각각 동일한 이름의 외부 output에 연결한다.

1. VHDL Code – Data Processor

```
39    component simple_alu
40       PORT (   op1, op2 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
41                sel      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
42                alu_out  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
43    end component;
44
45  BEGIN
46     RAM : asynch_ram
47        port map (data_in => alu_out_signal, address => MAR, wr => ram_load, data_out => ram_out, m1_out=>m1_out, m2_out=>m2_out);
48     ALU : simple_alu
49        port map (op1 => AC, op2 => ram_out, sel => alu_sel, alu_out => alu_out_signal) ;
50  PROCESS (mar_load, ac_load, input_sel, clk)
51  BEGIN
52
53     --AC
54     IF clk'EVENT AND clk = '1' THEN
55        IF ac_load = '1' THEN
56           AC <= mux;
57        END IF;
58     END IF;
59
60     --MAR
61     IF clk'EVENT AND clk = '1' THEN
62        IF mar_load = '1' THEN
63           MAR <= Mar_in;
64        END IF;
65     END IF;
67       --MUX
68     IF input_sel = '1' THEN
69        mux <= input;
70     ELSE
71        mux <= alu_out_signal;
72     END IF;
73  END PROCESS;
74  output <= AC;
75  data_check <= alu_out_signal;
76  mux_out <= mux;
77  mar_out <= MAR;
78 END data_processor;
```

② ALU

→ Op1은 AC의 output인 AC와 연결하고, op2는 RAM의 output인 ram_out과 연결한다.

→ sel은 alu_sel과 연결하고, alu_out은 ALU의 output인 alu_out_signal과 연결한다.

1. VHDL Code – Data Processor

```
39    component simple_alu
40       PORT (   op1, op2 : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
41                sel      : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
42                alu_out  : OUT STD_LOGIC_VECTOR (3 DOWNTO 0) );
43    end component;
44
45 BEGIN
46    RAM : asynch_ram
47       port map (data_in => alu_out_signal, address => MAR, wr => ram_load, data_out => ram_out, m1_out=>m1_out, m2_out=>m2_out);
48    ALU : simple_alu
49       port map (op1 => AC, op2 => ram_out, sel => alu_sel, alu_out => alu_out_signal) ;
50 PROCESS (mar_load, ac_load, input_sel, clk)
51 BEGIN
52
53    --AC
54    IF clk'EVENT AND clk = '1' THEN
55       IF ac_load = '1' THEN
56          AC <= mux;
57       END IF;
58    END IF;
59
60    --MAR
61    IF clk'EVENT AND clk = '1' THEN
62       IF mar_load = '1' THEN
63          MAR <= Mar_in;
64       END IF;
65    END IF;
67     --MUX
68     IF input_sel = '1' THEN
69        mux <= input;
70     ELSE
71        mux <= alu_out_signal;
72     END IF;
73  END PROCESS;
74  output <= AC;
75  data_check <= alu_out_signal;
76  mux_out <= mux;
77  mar_out <= MAR;
78 END data_processor;
```

→ AC는 Clock의 rising edge에서 ac_load가 1일 때, AC에 mux값을 반영해준다.

→ MAR은 Clock의 rising edge에서 mar_load가 1일 때, MAR에 MAR_in값을 반영해준다.

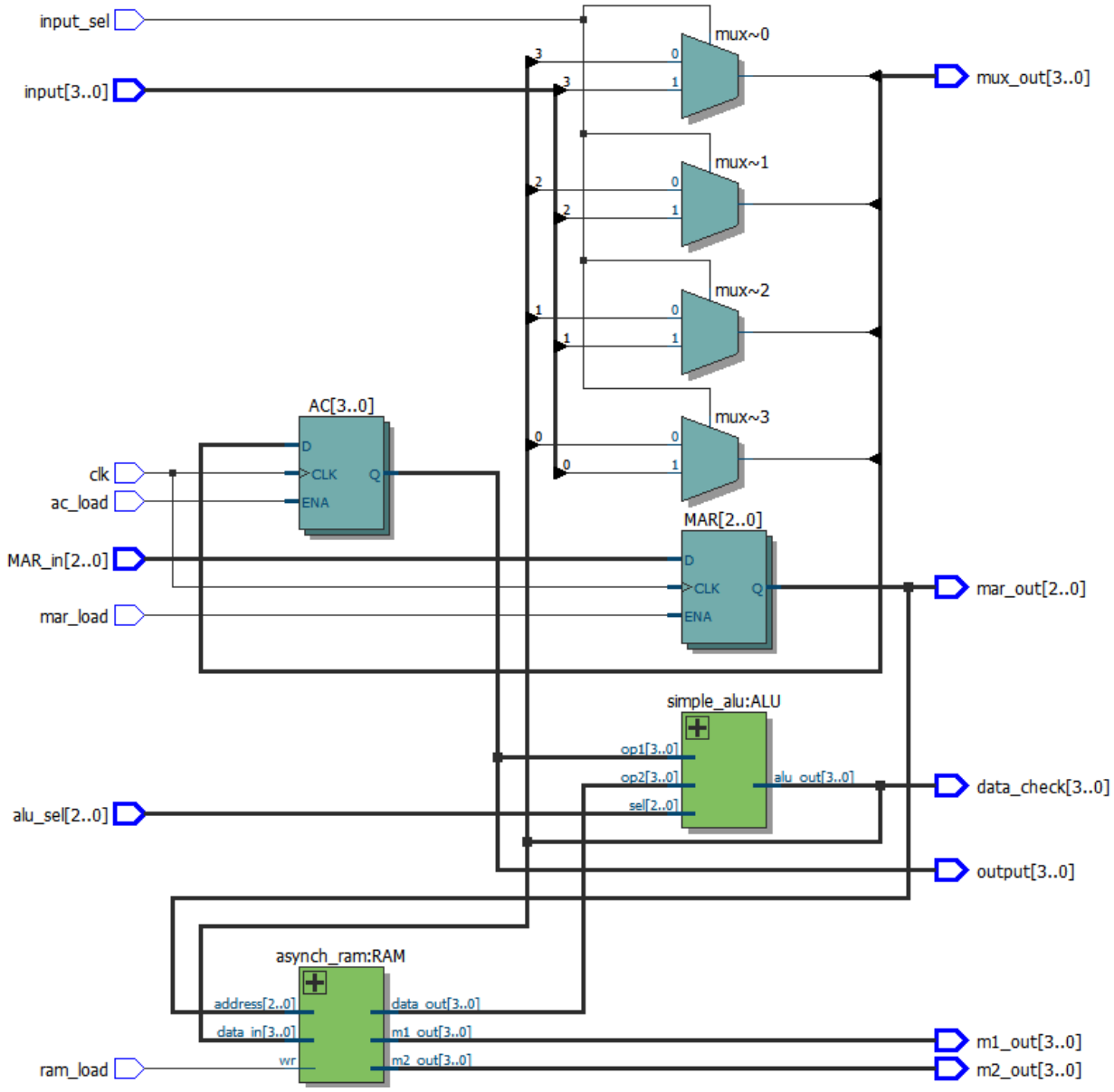
→ MUX는 inut_sel이 1일 때, mux에 input값을 넣고, 그 외에는 mux에 alu_sout_signal값을 넣어준다.

→ 그리고 output을 다음과 같이 뽑아 내부 Signal을 외부에서 볼 수 있게 해준다.

2. RTL Viewer

→ 다음은 위의 코드를 RTL Viewer로 나타낸 것이다.

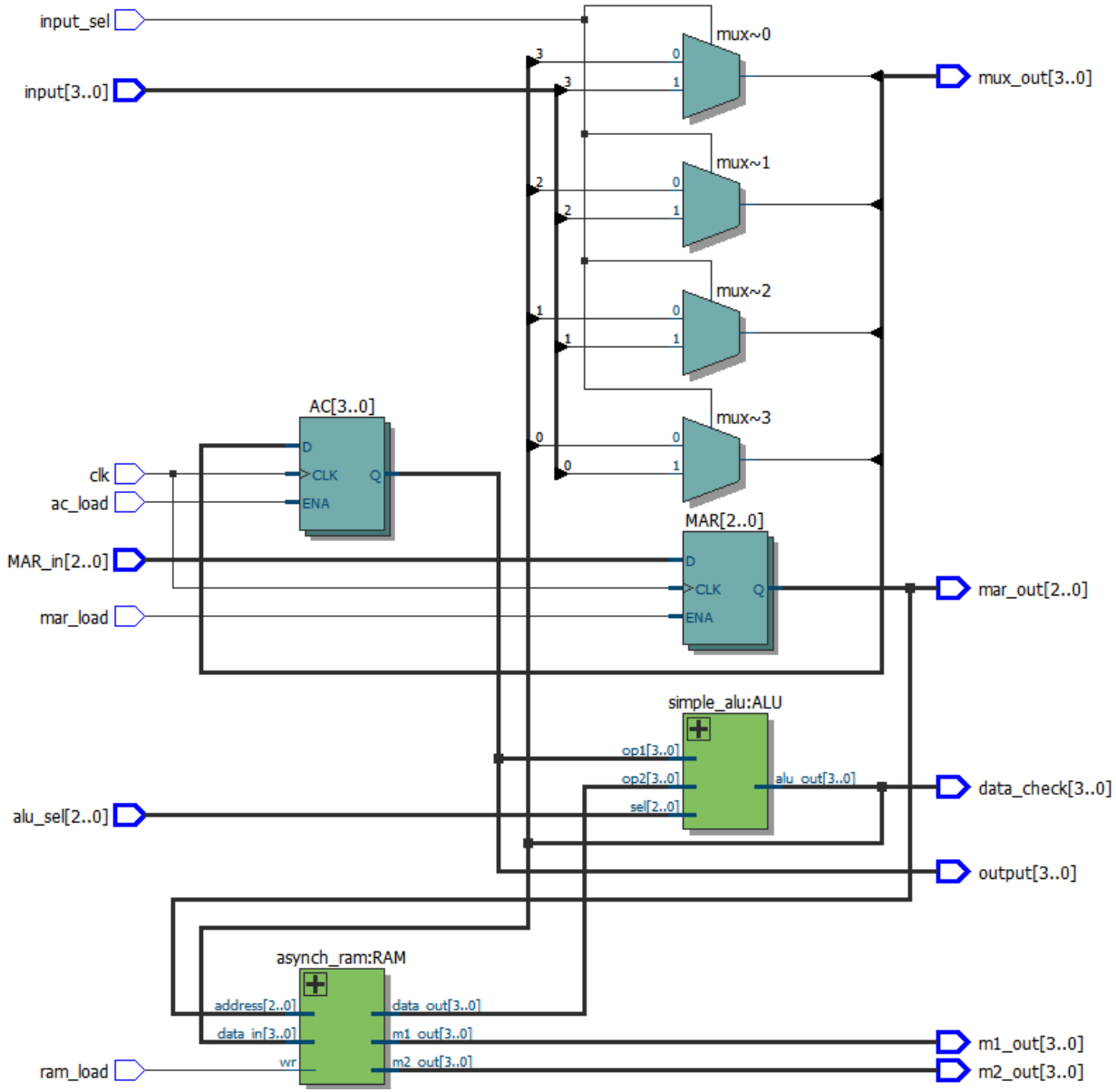
→ 이를 통해 내부 Signal을 확인하기 위해서 외부로 빼서 출력한 것을 볼 수 있다.



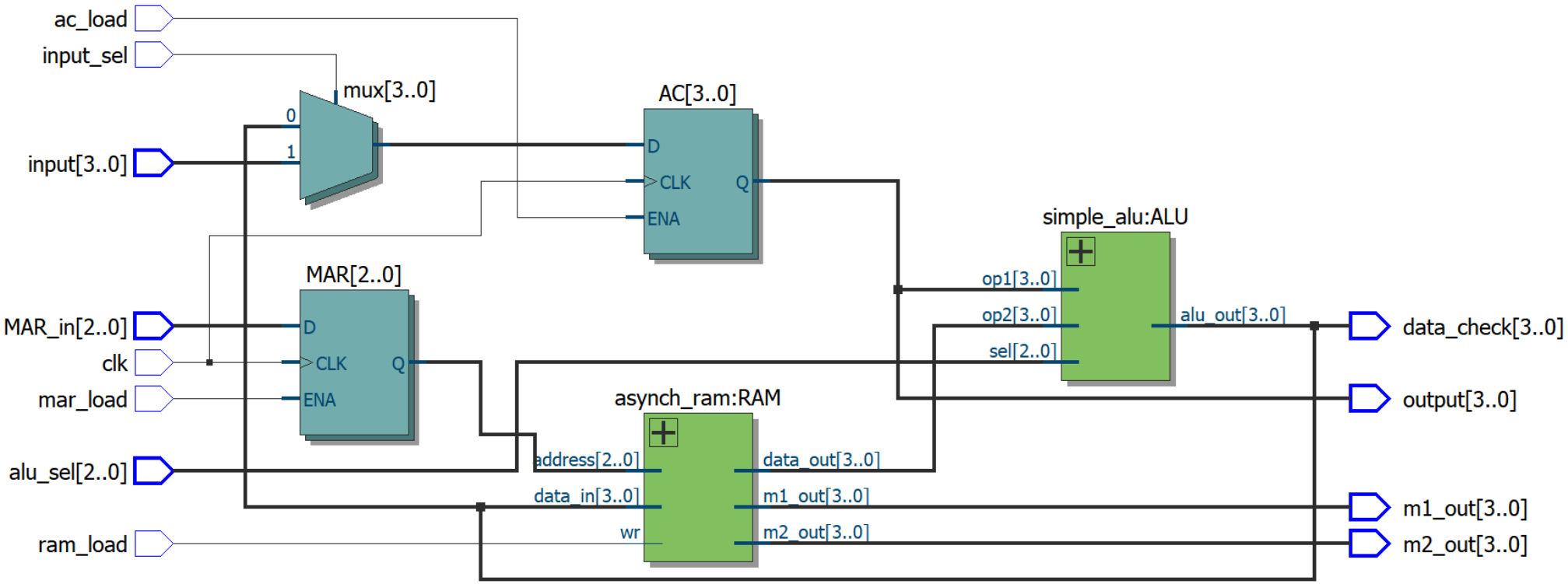
2. RTL Viewer

→ AC와 MAR은 각각 clk의 영향을 받고, RAM과 ALU는 component를 가지고 와서 instantication을 해준다.

→ 각각의 mux는 하나인데 이를 외부로 출력하려다 보니 1bit씩 표현되게 되었다.

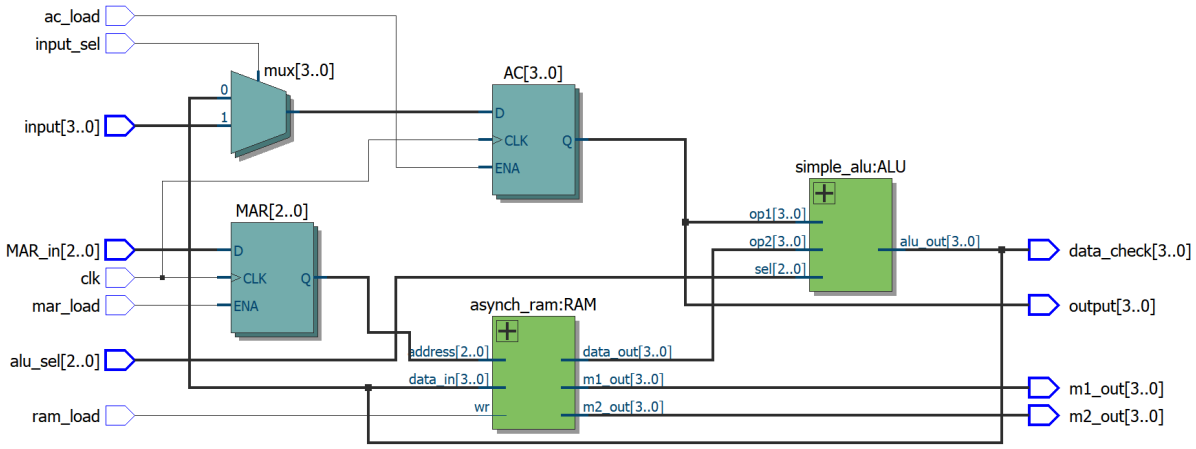


2. RTL Viewer



→ Mux와 MAR을 외부로 출력하지 않으면 다음과 같은 RTL Viewer가 나오게 된다.

2. RTL Viewer



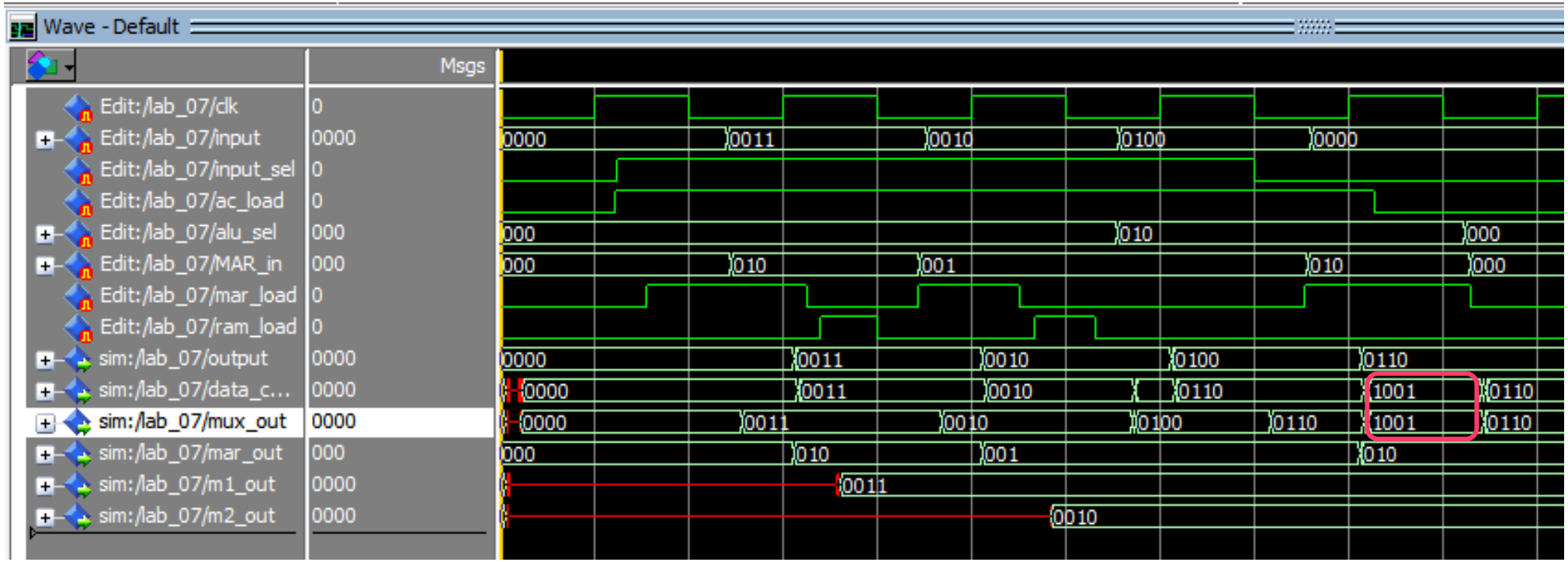
→ Input_sel이 1이면 mux에 input이 들어가고, 해당 input은 AC를 통해 ALU로 들어가게 된다. AC에서 ac_load가 1인 경우에 저장이 되고, ALU를 통해 밖으로 나온 input은 RAM에 들어가게 된다. MAR은 clock과 동기화 되었고, mar_load가 1일 때 MAR_in을 받고, 이를 RAM의 address로 보내게 된다. 따라서 RAM에서 해당 address에 Input값이 저장되는 구조를 볼 수 있다.

3. Simulation

- 다음의 동작을 순차적으로 수행하도록 simulation 하고 검증하라.
  - ac ← input (‘3’)
  - M[2] ← ac
  - ac ← input (‘2’)
  - M[1] ← ac
  - ac ← input (‘4’)
  - ac ← ac + M[1]
  - ac ← ac + M[2]

→ 주어진 순서를 토대로 Simulation을 진행하였다.

3. Simulation

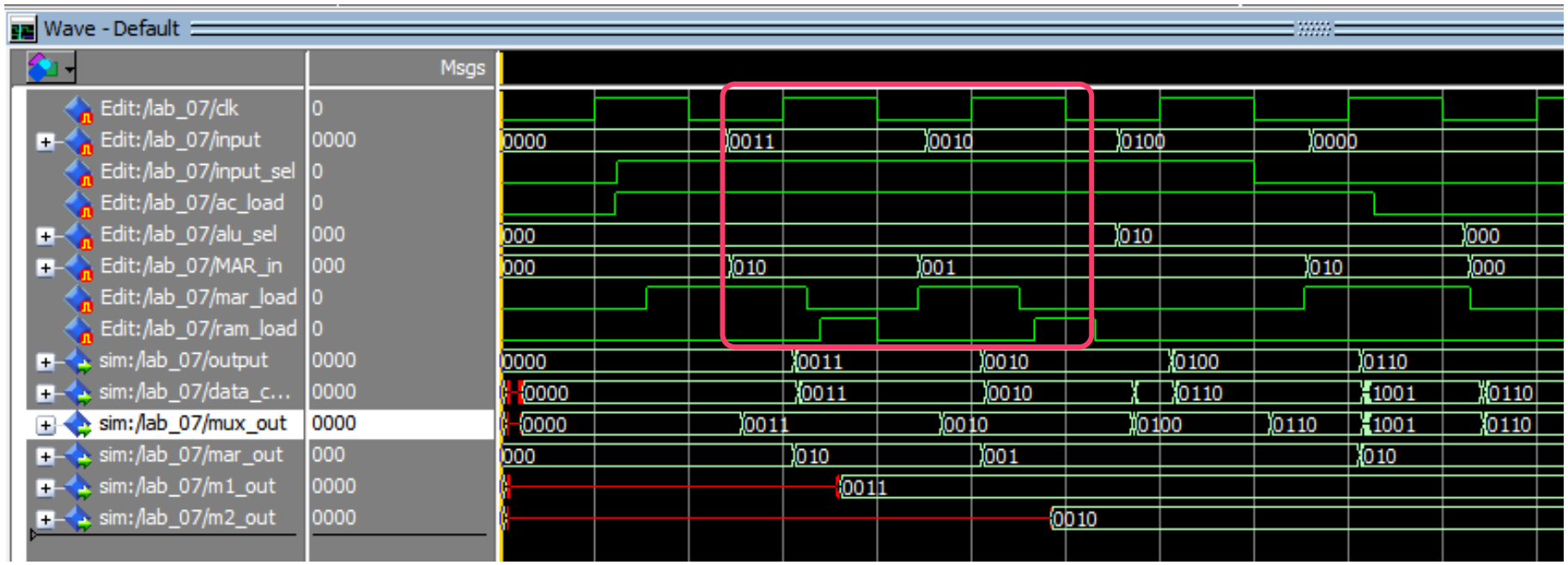


- ac ← input (‘3’)
- M[2] ← ac
- ac ← input (‘2’)
- M[1] ← ac
- ac ← input (‘4’)
- ac ← ac + M[1]
- ac ← ac + M[2]

→ 다음은 Simulation 결과를 나타낸 것이다.

→ 결과적으로 1001(9)가 나온 것을 확인할 수 있다.

3. Simulation

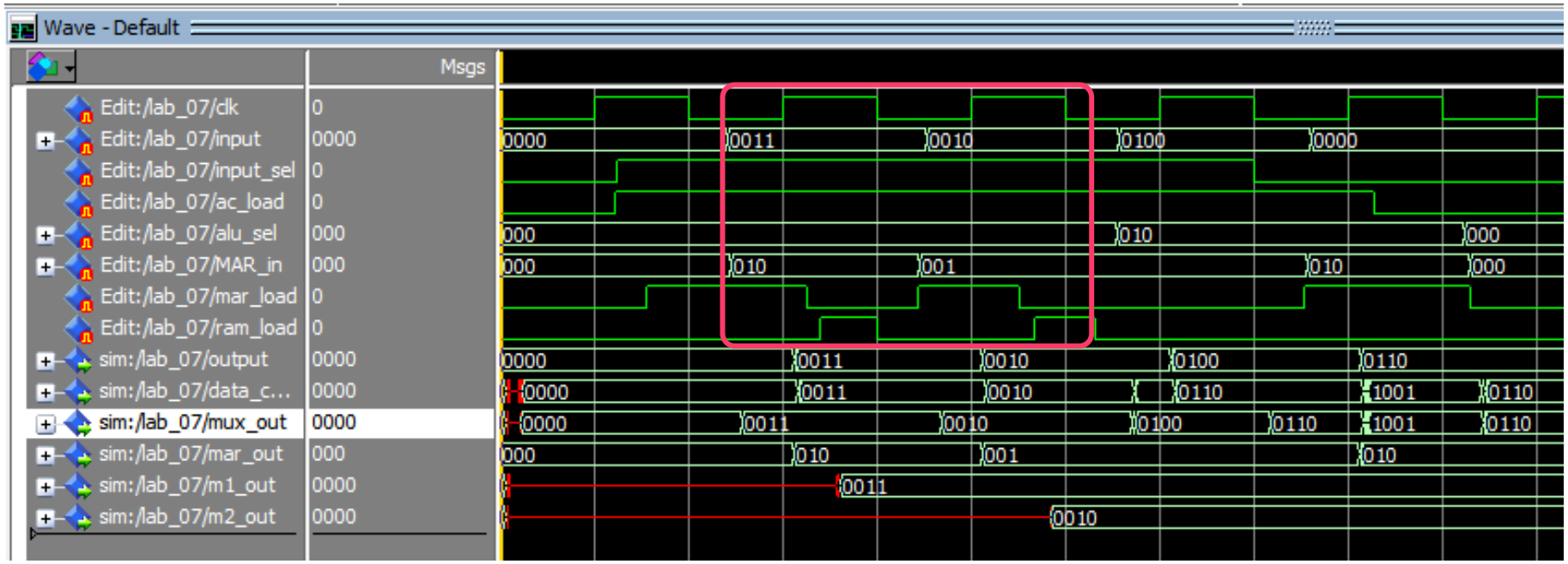


- ac ← input ('3')
- M[2] ← ac
- ac ← input ('2')
- M[1] ← ac

→ 이 부분 Simulation 설명

→ 해당 Simulation은 input이 Clock의 rising edge에서 0011(3)일 때, address 010(2)에 저장하였고, input이 0010(2)일 때, address 001(1)에 저장하였다. (mar_load가 1이고, 다음에 ram_load가 1이므로 MAR_in을 Clock의 rising edge에서 받아 해당 address에 Input을 write한 것이다.)

3. Simulation

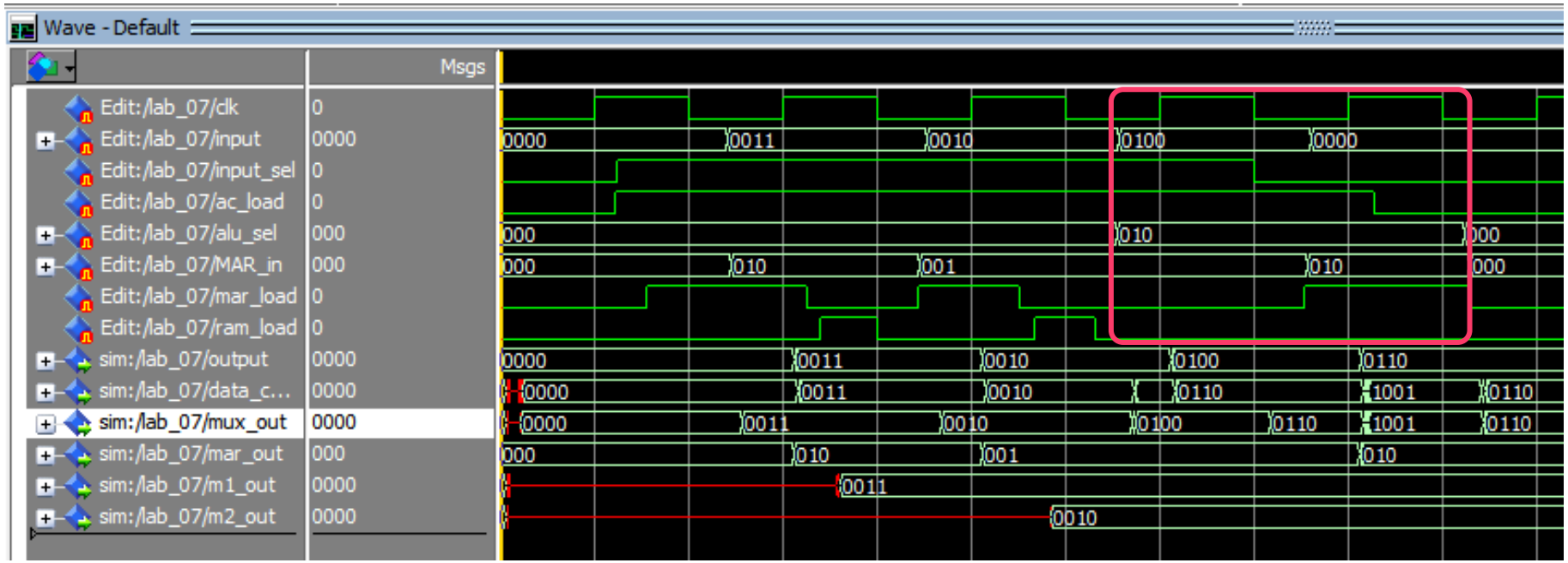


- ac ← input ('3')
- M[2] ← ac
- ac ← input ('2')
- M[1] ← ac

→ 이 부분 Simulation 설명

→ 해당 부분의 결과를 보면 output에는 ALU에 들어가기 전에 값이 반영되어 있고, data_check는 ALU에서 나간 값이 반영되어 나타났다. 또한, mux_out을 통해 해당 mux값을 알 수 있고, mar_out에 mar_in값이 반영되어 나타난 것을 알 수 있다. 그리고 m1_out과 m2_out을 통해 해당 address 위치에 각각 input값이 잘 저장된 것을 확인 할 수 있다.

3. Simulation

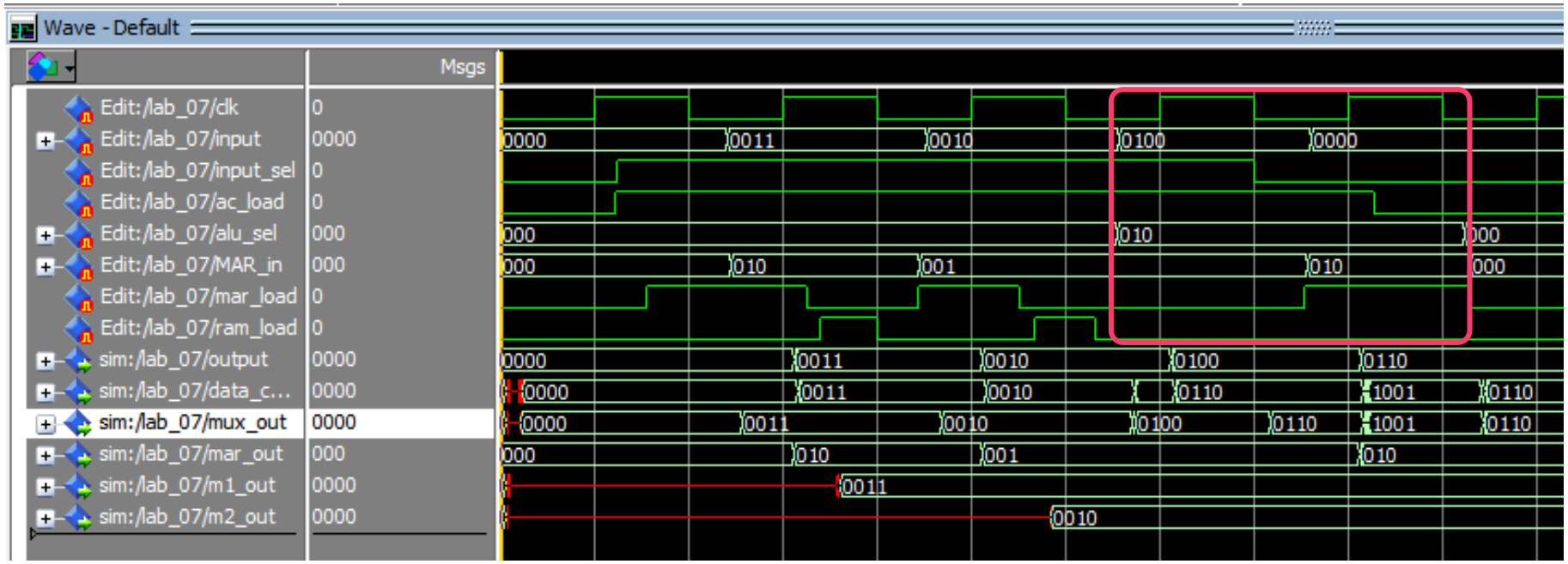


- ac ← input ('4')
- ac ← ac + M[1]
- ac ← ac + M[2]

→ 이 부분 Simulation 설명

→ input이 0100(4)일 때, address 001(1)에 있는 값과 더하였고, 다음 Clock에서 address 010(2)에 있는 값과 앞에서 더한 값의 결과를 더하였다. 따라서 더한 결과 1001(9)가 나오게 되었다. (ac_load를 다음 Clock까지 진행시켜 ac+M[1]값을 저장한 다음 address를 이동해서 해당 address에 있는 값인 M[2]와 ac+M[1]한 값을 더해주었다.)

3. Simulation



- ac ← input ('4')
- ac ← ac + M[1]
- ac ← ac + M[2]

→ 이 부분 Simulation 설명

→ 해당 부분의 결과를 보면 output에는 ALU에 들어가기 전에 값이 반영되어서 결과인 1011은 나타나지 않았고, data_check는 ALU에서 나간 값이 반영되었다. 이를 통해 최종 값이 1001(9)이라는 것을 알 수 있다. 또한, mux_out을 통해 해당 mux값을 알 수 있고, mar_out에 mar_load에 따라 mar_in값이 반영되어 나타난 것을 알 수 있다.

4. Discussion

→ 이번 과제는 특히 많은 시간이 소요되었다. 코드를 짤 때도 각각의 Signal들이 어떻게 연결되는지 하나하나 따로 살펴보아야 했고, 이를 연결해 줄 내부 Signal도 만들어야 했다. 처음에 연결을 할 때 내부 Signal을 최소화하고자 하는 마음에 연결을 하지 않은 부분이 생겨서 난감한 경우도 발생하였다. 결국에는 사용할 수 있는 모든 Signal을 사용하여 코드를 완성시켰다. 또한, 주어진 Diagram과 RTL Viewer를 관찰해가면서 하나씩 연결해 나가는 과정을 거쳐서 다음과 같은 코드를 완성시켰다. 다음으로 Simulation을 할 때 역시 많은 어려움이 있었다. 일단 저장하는 것까지 Simulation이 되는 것까지도 많은 시행착오가 있었고, 여러 번의 Simulation을 통해 이를 해결할 수 있었다.

4. Discussion

→ 마찬가지로 더하는 과정에서도 많은 시행착오가 있었는데, 특히 M[2]를 더하는 과정에서 여러 시행착오를 겪게 되었다. M[1]이 더해지는 것까진 되었지만, 이 다음인 M[2]를 더하지 못하였다. 결국 이는 Clock과 ac_load의 관계를 명확하게 파악하지 못하면서 M[1]과 ac가 더해진 값을 저장해주지 않고 넘어가면서 생긴 문제라는 것을 알 수 있었고, 이를 해결 할 수 있었다.

→ 이번 Simulation은 이해하는 것부터 시작해서 코드를 짜고 Simulation을 돌리기 까지 많은 어려움이 있었고, 이를 해결하기 위해서 많은 시간이 걸려서 어려웠다. 하지만, 이를 해결할 때마다 뿌듯함도 느낄 수 있었던 실습이었다.