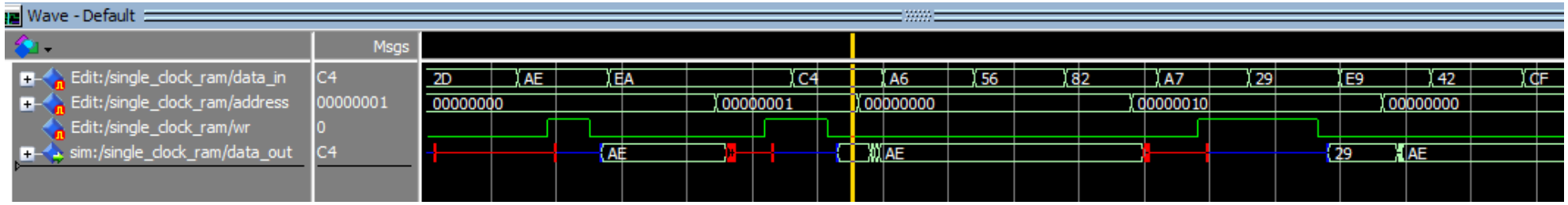
Mini-Lab 06.

RAM 분석(Simulation을 통한 동작 분석)

201810800 이혜인

06-1 Async RAM

- 다음 page의 VHDL code를 이용하여 Async, Dual-port, Single-address RAM을 구현하라.
  - Async, Dual-port, Single-address란 어떤 의미인가?
- RTL viewer의 block diagram을 분석하고 설명하라.
- 적당한 input으로 Gate-level simulation을 통해 timing 특성을 분석하고 설명하라. (다음 page의 simulation example을 참고하라.)




```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.numeric_std.all;

ENTITY asynch_ram IS
 PORT (          data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                 address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
                 wr: IN STD_LOGIC;
                 data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
END asynch_ram;

ARCHITECTURE rtl OF asynch_ram IS
 TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
 SIGNAL ram_block: MEM;
BEGIN
 PROCESS (wr, data_in, address)
 BEGIN
   IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
                      data_out <= (others => 'Z');
   else  data_out <= ram_block(to_integer(unsigned(address)));
  END IF;
 END PROCESS;
END rtl;
```

06-2 Dual clock RAM

- 다음 page의 VHDL code를 이용하여 Dual port, dual address, dual clock RAM을 구현하라.
  - dual port, dual address, dual clock이란 어떤 의미인가?
- RTL viewer의 block diagram을 분석하고 설명하라.
- 적당한 input으로 Gate-level simulation을 통해 timing 특성을 분석하고 설명하라.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

ENTITY dual_clock_ram IS
 PORT (
 clock1, clock2: IN STD_LOGIC;
 data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
 write_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
 read_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
 wr: IN STD_LOGIC;
 data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
 );
END dual_clock_ram;
```

```
ARCHITECTURE rtl OF dual_clock_ram IS
 TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
 SIGNAL ram_block: MEM;
 SIGNAL read_address_reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
 PROCESS (clock1)
 BEGIN
   IF (clock1'event AND clock1 = '1') THEN
     IF (wr = '1') THEN
         ram_block(to_integer(unsigned(write_address))) <= data_in;
     END IF;
   END IF;
 END PROCESS;
 PROCESS (clock2)
 BEGIN
 IF (clock2'event AND clock2 = '1') THEN
   data_out <= ram_block(to_integer(unsigned(read_address_reg)));
   read_address_reg <= read_address;
 END IF;
 END PROCESS;
END rtl;
```

06-1 Async RAM

1. VHDL Code – Async, Dual-port, Single-address의 의미

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY lab_06_1 IS
6    PORT (  data_in  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            address  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8            wr       : IN STD_LOGIC;
9            data_out : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
10  END lab_06_1;
11
12  ARCHITECTURE rtl OF lab_06_1 IS
13   TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
14   SIGNAL ram_block: MEM;
15  BEGIN
16   PROCESS (wr, data_in, address)
17   BEGIN
18     IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
19        data_out <= (others => 'Z');
20    else  data_out <= ram_block(to_integer(unsigned(address)));
21    END IF;
22   END PROCESS;
23  END rtl;
```

→ 다음 VHDL Code를 보면 input이 각각 data_in 8bit, address 8bit, wr (write를 의미)로 정의되어 있고, output은 data_out 8bit로 정의되어 있다.

→ Async : 해당 코드는 Clock이 없어 Clock의 영향을 받지 않으므로 이는 Asynchronous한 RAM이다.

06-1 Async RAM

1. VHDL Code – Async, Dual-port, Single-address의 의미

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  ENTITY lab_06_1 IS
6   PORT (  data_in  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7           address  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           wr       : IN STD_LOGIC;
9           data_out : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
10 END lab_06_1;
11
12 ARCHITECTURE rtl OF lab_06_1 IS
13  TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
14  SIGNAL ram_block: MEM;
15 BEGIN
16  PROCESS (wr, data_in, address)
17  BEGIN
18    IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
19       data_out <= (others => 'Z');
20    else  data_out <= ram_block(to_integer(unsigned(address)));
21   END IF;
22  END PROCESS;
23 END rtl;
```

→ Dual-port : 이와 같이 data_in, data_out 으로 read bus와 write bus가 구분되어 있는 것을 Dual-port라고 한다.

→ Single-address : 다음과 같이 하나의 address로 이루어져서 Read와 Write를 하나의 address line으로 지시하는 것을 Single-address라고 한다.

06-1 Async RAM

1. VHDL Code – Async, Dual-port, Single-address의 의미

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY lab_06_1 IS
6    PORT (  data_in  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            address  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8            wr       : IN STD_LOGIC;
9            data_out : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
10  END lab_06_1;
11
12  ARCHITECTURE rtl OF lab_06_1 IS
13   TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
14   SIGNAL ram_block: MEM;
15  BEGIN
16   PROCESS (wr, data_in, address)
17   BEGIN
18     IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
19        data_out <= (others => 'Z');
20    else  data_out <= ram_block(to_integer(unsigned(address)));
21    END IF;
22   END PROCESS;
23  END rtl;
```

→ 다음 Code는 MEMORY의 Data Type을 8bit word를 256개 쌓은 Array의 형태로 규정하였고, 'ram_block'이라는 이름의 MEMORY를 사용한다.

06-1 Async RAM

1. VHDL Code – Async, Dual-port, Single-address의 의미

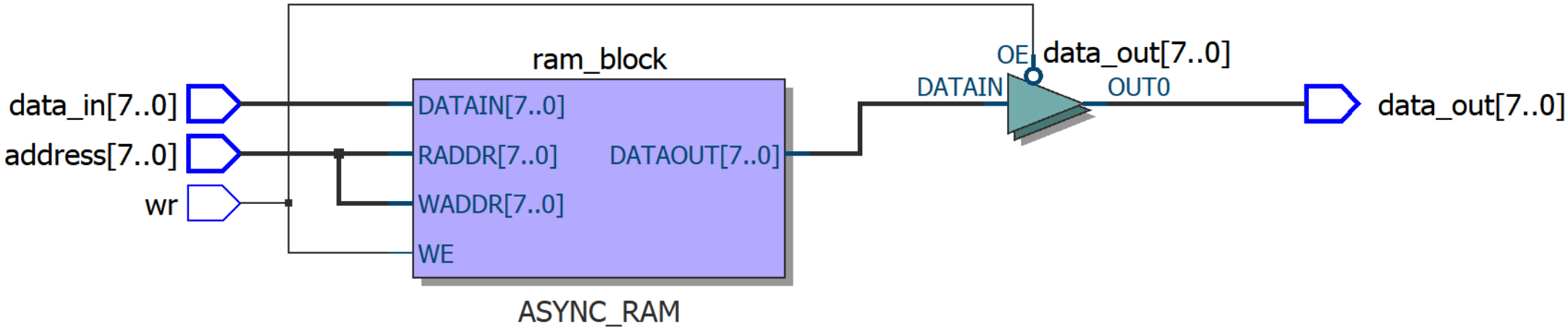
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  ENTITY lab_06_1 IS
6   PORT (  data_in  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7           address  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           wr       : IN STD_LOGIC;
9           data_out : OUT STD_LOGIC_VECTOR (7 DOWNTO 0) );
10 END lab_06_1;
11
12 ARCHITECTURE rtl OF lab_06_1 IS
13  TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
14  SIGNAL ram_block: MEM;
15 BEGIN
16  PROCESS (wr, data_in, address)
17  BEGIN
18    IF (wr = '1') THEN ram_block(to_integer(unsigned(address))) <= data_in;
19       data_out <= (others => 'Z');
20    else  data_out <= ram_block(to_integer(unsigned(address)));
21   END IF;
22  END PROCESS;
23 END rtl;
```

→ Process는 Clock없이 진행되며, wr이 1인 경우, ram_block의 address에 data_in을 input을 write한다. 그리고 data _out에 'Z'를 넣어 이를 끊어준다.

→ wr이 1이 아닌 경우(wr이 0인 경우), ram_block의 지정된 address에 해당되는 내용을 data_out에 실어서 read-out해준다.

06-1 Async RAM

2. RTL Viewer

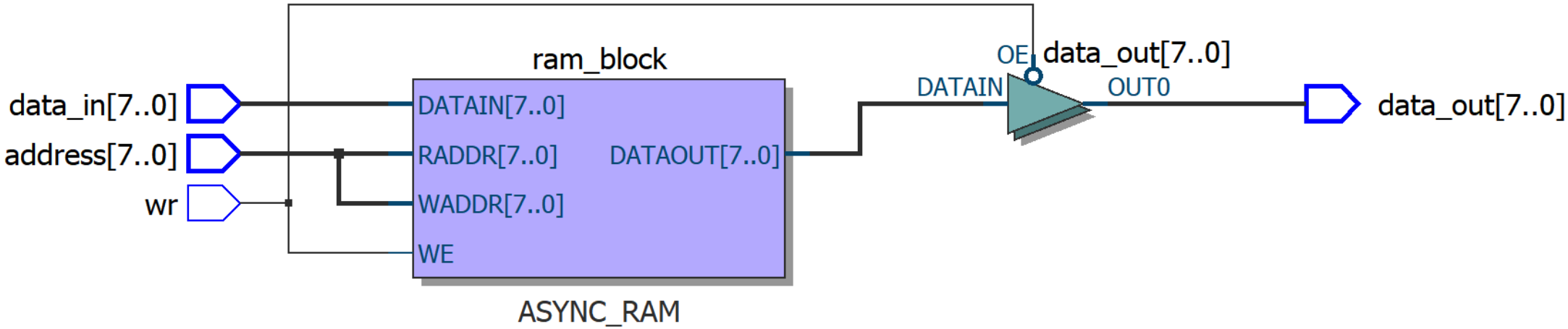


→ 다음은 위의 VHDL Code를 RTL Viewer로 나타낸 것이다.

→ 이는 input으로 data_in, address, wr이 들어오고, output은 data_out이다. 이를 통해 dual-port, Single-address를 사용했다는 것을 알 수 있고, Clock에 동기화되지 않았으므로 해당 ram_block은 Asynchronous RAM인 것을 알 수 있다.

06-1 Async RAM

2. RTL Viewer

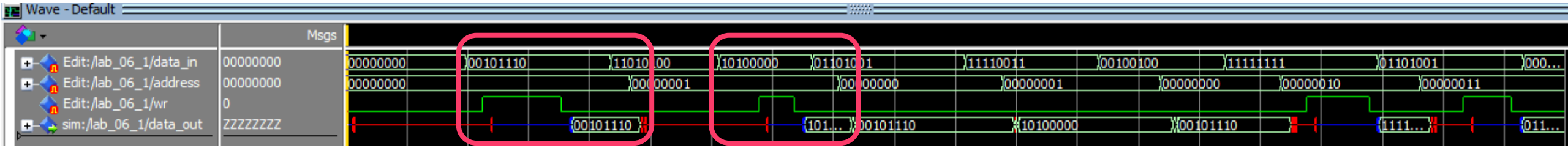


→ 그리고 wr이 1인 경우에는 RAM에 해당 address에 write를 해주는 것을 알 수 있고, 0인 경우에는 data_out에서 지정된 내용을 read-out해주는 것을 알 수 있다.

06-1 Async RAM

3. Gate-level Simulation

① Binary Number



→ Input인 data_in, address, wr는 임의로 설정하였다.

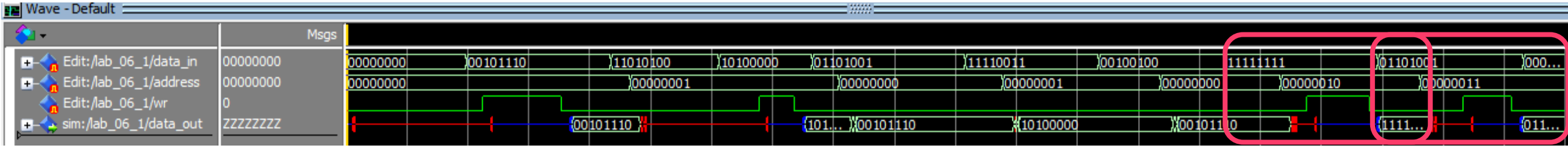
① Data_in이 00101110이고, adress가 0000000일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

② Data_in이 10100000이고, adress가 0000001일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

06-1 Async RAM

3. Gate-level Simulation

① Binary Number



③ Data_in이 11111111이고, adress가 00000010일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

④ Data_in이 01101001이고, adress가 0000011일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

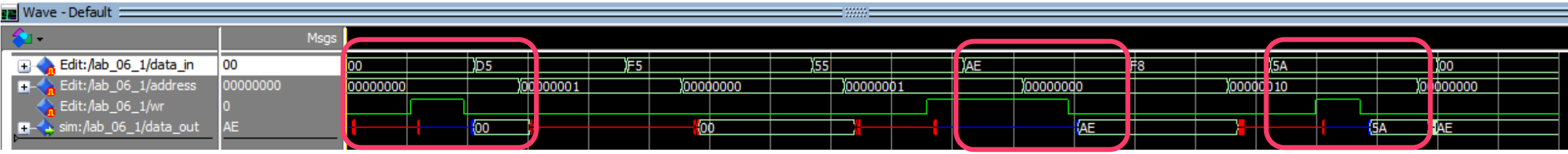
→ 이 외에 data_out은 해당 address 값이나 Z값(address안에 값이 없는 경우)이 출력된 상태이다.

→ 빨간 부분은 값이 없는 상태, 파란 부분은 write하고 있는 상태이다.

06-1 Async RAM

3. Gate-level Simulation

② Hexadecimal Number



→ Input인 data_in, address, wr는 임의로 설정하였다. – 2진수와 동일하게 진행하였고, 강의자료를 참고하여 16진수로 Simulation을 돌려보았다.

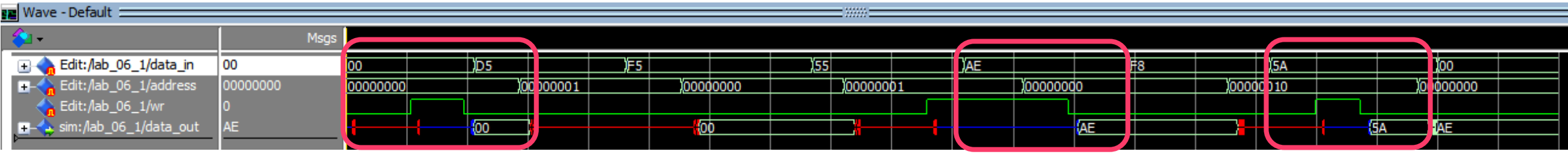
① Data_in이 00이고, adress가 0000000일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

② Data_in이 AE이고, adress가 0000001일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

06-1 Async RAM

3. Gate-level Simulation

② Hexadecimal Number



③ Data_in이 5A이고, adress가 00000010일 때, wr(write)가 1이 되었으므로 해당 address에 data가 write된다. 그리고 이 값은 write가 0이 되고 적당한 propagation delay 이후에 data_out(read-out)이 된다.

→ 이 외에 data_out은 해당 address 값이나 Z값(address안에 값이 없는 경우)이 출력된 상태이다.

→ 빨간 부분은 값이 없는 상태, 파란 부분은 write하고 있는 상태이다.

06-2 Dual Clock RAM

1. VHDL Code – dual port, dual address, dual clock의 의미

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY lab_06_2 IS
6    PORT ( clock1, clock2: IN STD_LOGIC;
7           data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           write_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
9           read_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
10          wr: IN STD_LOGIC;
11          data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
12   );
13  END lab_06_2;
14  ARCHITECTURE rtl OF lab_06_2 IS
15   TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
16   SIGNAL ram_block: MEM;
17   SIGNAL read_address_reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
18  BEGIN
19   PROCESS (clock1)
20   BEGIN
21     IF (clock1'event AND clock1 = '1') THEN
22       IF (wr = '1') THEN
23           ram_block(to_integer(unsigned(write_address))) <= data_in;
24       END IF;
25     END IF;
26   END PROCESS;
27   PROCESS (clock2)
28   BEGIN
29   IF (clock2'event AND clock2 = '1') THEN
30     data_out <= ram_block(to_integer(unsigned(read_address_reg)));
31     read_address_reg <= read_address;
32   END IF;
33   END PROCESS;
34  END rtl;
```

→ 다음 VHDL Code를 보면 input이 각각 clock1, clock2와 data_in 8bit, write address, read address 각각 8bit, wr (write를 의미)로 정의되어 있고, output은 data_out 8bit로 정의되어 있다.

→ Dual-port : 이와 같이 data_in, data_out으로 read bus와 write bus가 구분되어 있는 것을 Dual-port라고 한다.

06-2 Dual Clock RAM

1. VHDL Code – dual port, dual address, dual clock의 의미

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY lab_06_2 IS
6    PORT ( clock1, clock2: IN STD_LOGIC;
7           data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           write_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
9           read_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
10          wr: IN STD_LOGIC;
11          data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
12   );
13  END lab_06_2;
14  ARCHITECTURE rtl OF lab_06_2 IS
15   TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
16   SIGNAL ram_block: MEM;
17   SIGNAL read_address_reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
18  BEGIN
19   PROCESS (clock1)
20   BEGIN
21     IF (clock1'event AND clock1 = '1') THEN
22       IF (wr = '1') THEN
23           ram_block(to_integer(unsigned(write_address))) <= data_in;
24       END IF;
25     END IF;
26   END PROCESS;
27   PROCESS (clock2)
28   BEGIN
29   IF (clock2'event AND clock2 = '1') THEN
30     data_out <= ram_block(to_integer(unsigned(read_address_reg)));
31     read_address_reg <= read_address;
32   END IF;
33   END PROCESS;
34  END rtl;
```

→ Dual address : 다음과 같이 write_address와 read_address로 각각의 기능이 구분되어 있는 것을 dual address라고 한다.

→ Dual clock : 다음과 같이 clock1과 clock2와 같이 2개의 Clock의 영향을 받는 것을 Dual clock이라고 한다.또한, 이는 Clock의 영향을 받으므로 Synchronous RAM이다.

06-2 Dual Clock RAM

1. VHDL Code – dual port, dual address, dual clock의 의미

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY lab_06_2 IS
6    PORT ( clock1, clock2: IN STD_LOGIC;
7           data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           write_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
9           read_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
10          wr: IN STD_LOGIC;
11          data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
12   );
13  END lab_06_2;
14  ARCHITECTURE rtl OF lab_06_2 IS
15   TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
16   SIGNAL ram_block: MEM;
17   SIGNAL read_address_reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
18  BEGIN
19   PROCESS (clock1)
20   BEGIN
21     IF (clock1'event AND clock1 = '1') THEN
22       IF (wr = '1') THEN
23           ram_block(to_integer(unsigned(write_address))) <= data_in;
24       END IF;
25     END IF;
26   END PROCESS;
27   PROCESS (clock2)
28   BEGIN
29   IF (clock2'event AND clock2 = '1') THEN
30     data_out <= ram_block(to_integer(unsigned(read_address_reg)));
31     read_address_reg <= read_address;
32   END IF;
33   END PROCESS;
34  END rtl;
```

→ 다음 Code는 MEMORY의 Data Type을 8bit word를 256개 쌓은 Array의 형태로 규정하였고, 'ram_block'이라는 이름의 MEMORY를 사용한다.

→ 또한, read_address_reg는 read_address와 clock2가 rising edge일 때 받아서 RAM으로 연결해주는 Register이다.

06-2 Dual Clock RAM

1. VHDL Code – dual port, dual address, dual clock의 의미

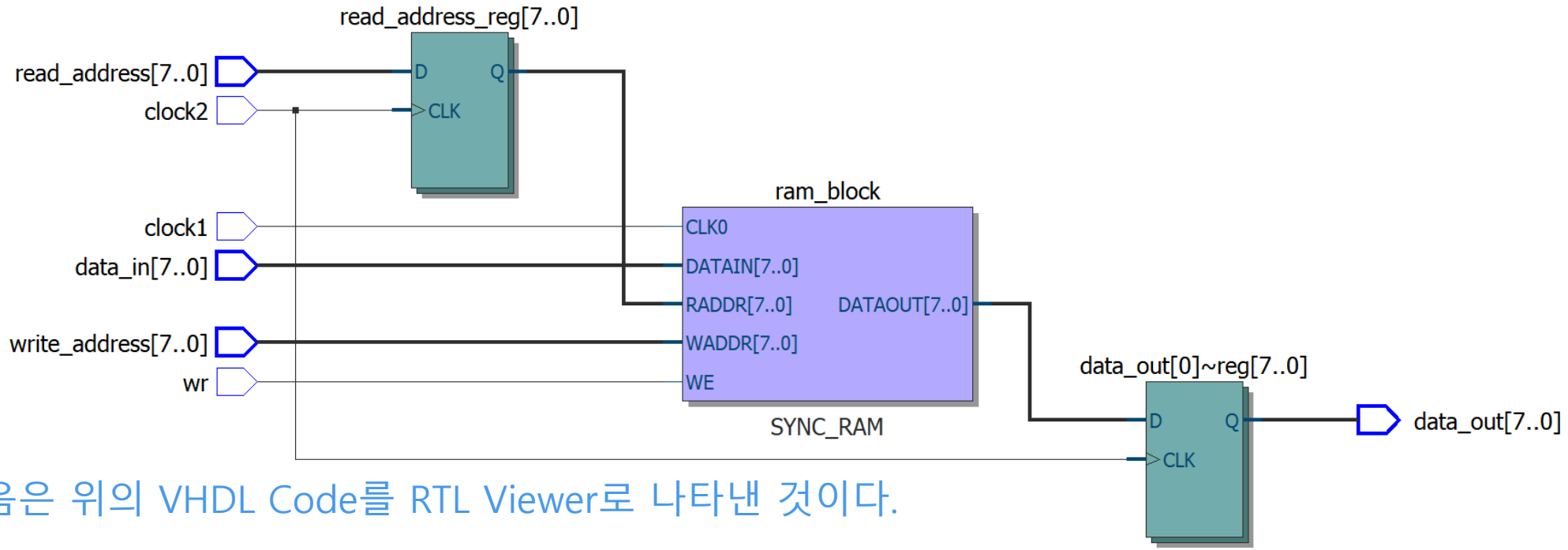
```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3   USE ieee.numeric_std.all;
4
5   ENTITY lab_06_2 IS
6    PORT ( clock1, clock2: IN STD_LOGIC;
7           data_in: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
8           write_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
9           read_address: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
10          wr: IN STD_LOGIC;
11          data_out: OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
12   );
13  END lab_06_2;
14  ARCHITECTURE rtl OF lab_06_2 IS
15   TYPE MEM IS ARRAY(0 TO 255) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
16   SIGNAL ram_block: MEM;
17   SIGNAL read_address_reg : STD_LOGIC_VECTOR (7 DOWNTO 0);
18  BEGIN
19   PROCESS (clock1)
20   BEGIN
21     IF (clock1'event AND clock1 = '1') THEN
22       IF (wr = '1') THEN
23           ram_block(to_integer(unsigned(write_address))) <= data_in;
24       END IF;
25     END IF;
26   END PROCESS;
27   PROCESS (clock2)
28   BEGIN
29   IF (clock2'event AND clock2 = '1') THEN
30     data_out <= ram_block(to_integer(unsigned(read_address_reg)));
31     read_address_reg <= read_address;
32   END IF;
33   END PROCESS;
34  END rtl;
```

→ Process(clock1)에서 clock1이 rising edge일 때, wr이 1이면, ram_block의 address에 data_in을 write한다.

→ Process(clock2)에서 clock2가 rising edge일 때, ram_block의 지정된 address에 해당되는 내용을 data_out에 실어서 read-out 해준다. 그리고 read_address_reg에 read_address를 받아온다.

06-2 Dual Clock RAM

2. RTL Viewer



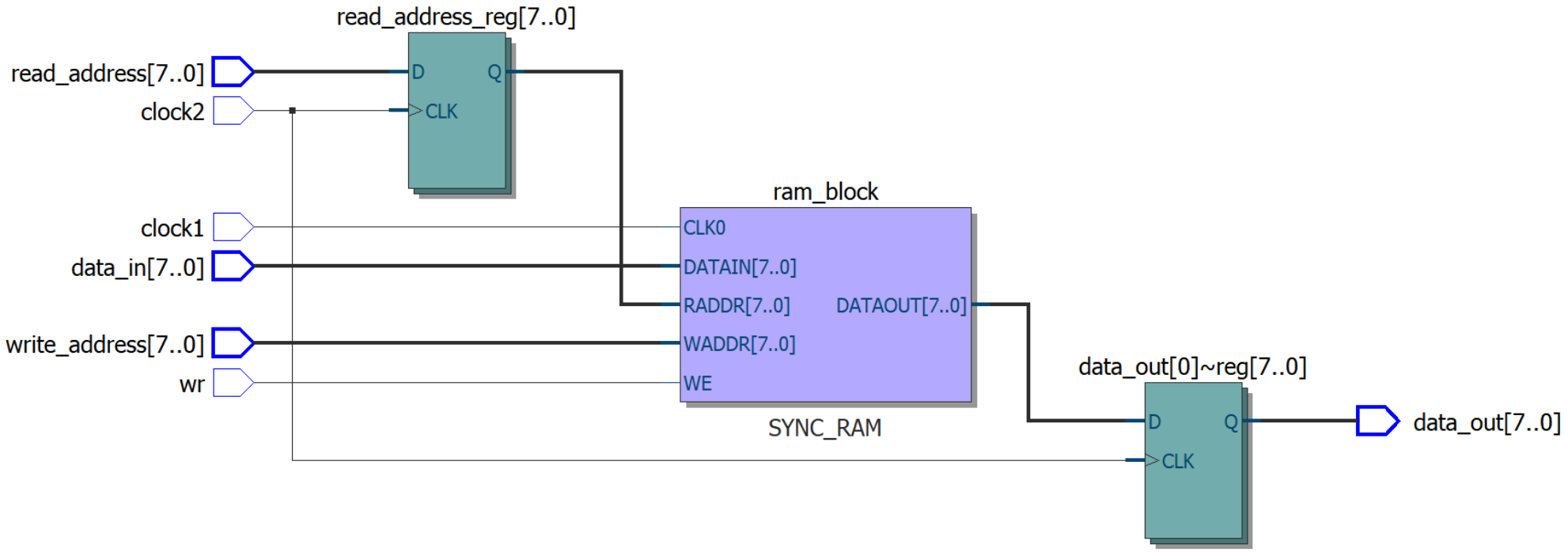
→ 다음은 위의 VHDL Code를 RTL Viewer로 나타낸 것이다.

→ 이는 input으로 clock1, clock2, data_in, read_address, write_address, wr이 들어오고, output은 data_out이다.

→ 이를 통해 dual-port, dual-address 그리고 dual clock을 사용했다는 것을 알 수 있다.

06-2 Dual Clock RAM

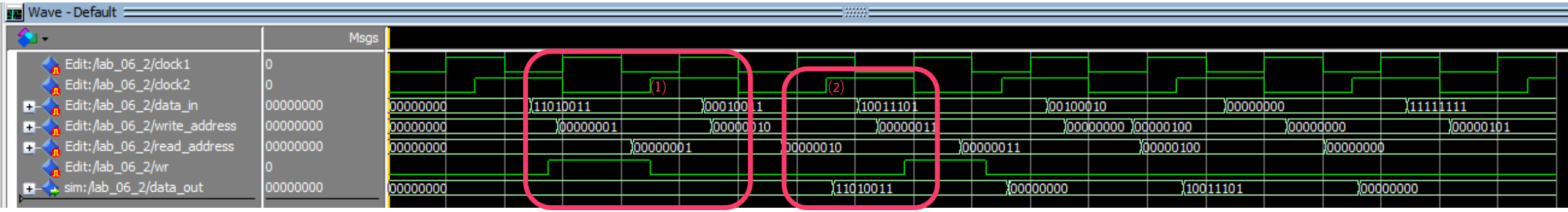
2. RTL Viewer



→ 또한, clock1은 SYNC_RAM에 동기화되고 clock2는 read_address_reg와 data_out에 동기화되므로, 해당 ram_block은 Synchronous RAM인 것을 알 수 있다.

06-2 Dual Clock RAM

3. Gate-level Simulation

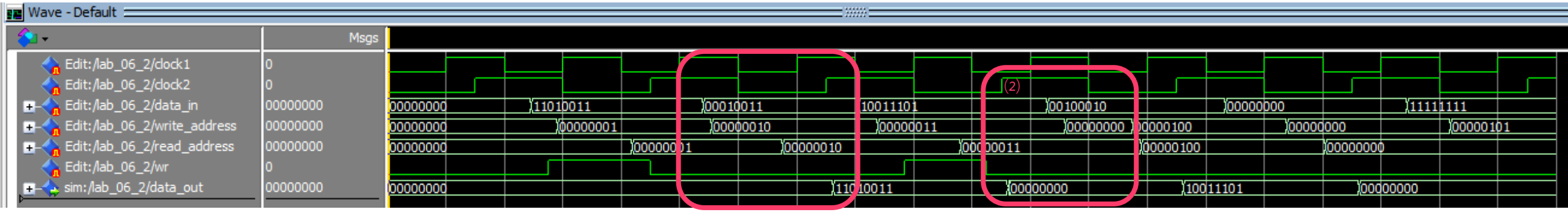


→ Input인 data_in, address, wr는 임의로 설정하였다.

① clock1이 rising edge일 때, wr이 1이 되었고, data_in은 11010011로 하고, write_address는 00000001로 해서 해당 address에 data가 write 된다. 그리고 write된 값은 clock2의 rising edge에서 out이 되는데, 먼저 (1) 에서는 read_address_reg에 read_address를 받아오고, 다음 clock2 rising edge, 즉 (2) 에서 data_out을 통해 read-out이 된다.

06-2 Dual Clock RAM

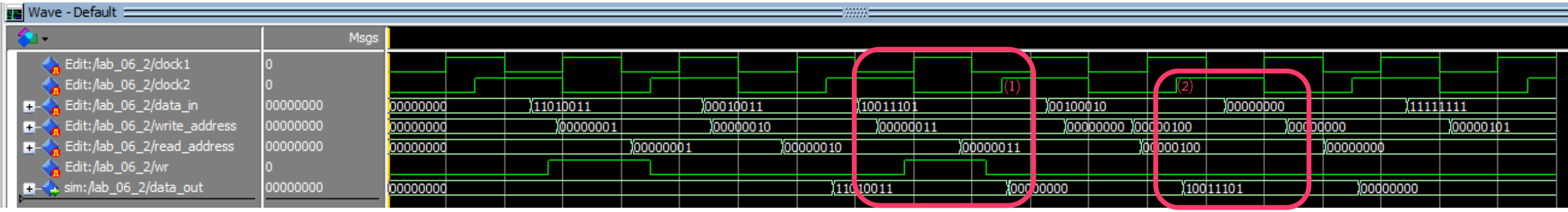
3. Gate-level Simulation



② clock1이 rising edge일 때, data_in은 00010011로 하고, write_address는 00000010으로 입력했지만, 해당 rising edge에서는 wr이 0이므로 write 되지 않는다. 따라서 write된 값이 없으므로 clock2의 rising edge에서 out될 값이 없어서 해당 출력 부분인 (2) 에서는 00000000이 read-out되게 된다.

06-2 Dual Clock RAM

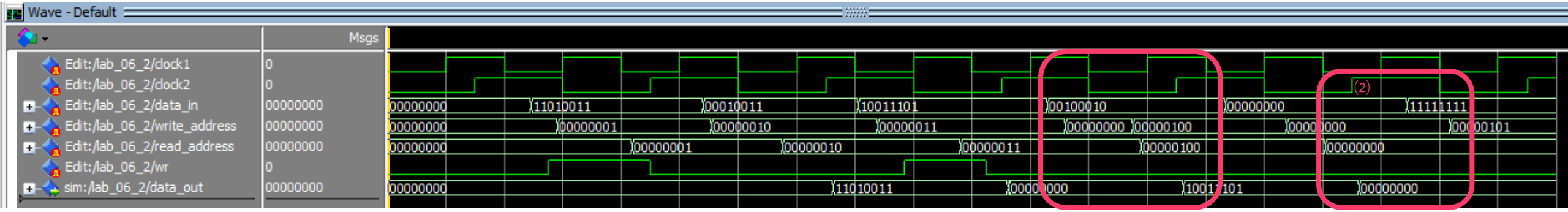
3. Gate-level Simulation



③ clock1이 rising edge일 때, wr이 1이 되었고, data_in은 10011101로 하고, write_address는 00000011로 해서 해당 address에 data가 write 된다. 그리고 write된 값은 clock2의 rising edge에서 out이 되는데, 먼저 (1) 에서는 read_address_reg에 read_address를 받아오고, 다음 clock2 rising edge, 즉 (2) 에서 data_out을 통해 read-out이 된다.

06-2 Dual Clock RAM

3. Gate-level Simulation



④ clock1이 rising edge일 때, data_in은 00100010로 하고, write_address는 00000100으로 입력했지만, 해당 rising edge에서는 wr이 0이므로 write 되지 않는다. 따라서 write된 값이 없으므로 clock2의 rising edge에서 out될 값이 없어서 해당 출력 부분인 (2) 에서는 00000000이 read-out되게 된다.

4. Discussion

→ 주어진 VHDL Code와 이를 실행해서 나타난 RTL Viewer를 해석하면 해당 Simulation이 어떻게 나오는지 알 수 있어서 편리하였다. 또한, 이번 실습을 통해 VHDL Code로 RAM을 나타내는 법을 알게 되었고, ROM과 RAM의 차이를 확실하게 알 수 있었다.

→ 또한, dual-clock을 처음 Simulation 했을 때는 하나의 Clock이 밀려나와 처음에 당황했지만, RTL Viewer를 보면서 차근차근 Simulation을 해석해보았을 때, register를 거쳐오느라 한 Clock 다음에 나오는 것을 알 수 있었다. 그리고 dual-address에 대해서도 Code와 RTL을 통해서 이해하고, 무리없이 Simulation을 진행할 수 있었다.