Mini-Lab 04.

ROM을 이용한 Combinational Logic 설계

201810800 이혜인

- 임의의 3-variable function F(A,B,C)의 진리표를 그려라.
- 해당 function F를 다음 구성 요소를 활용하여 구현하라.
  - Megawizard Plug-in manager를 이용하여 구현한 ROM
  - mif 파일
- 진리표 대로 구현되었는지 simulation을 통해 검증하라.
  - 강의 자료의 code를 이용하면 불필요한 Clock 시그널이 포함될 수 있다. 적당히 처리하라

Index

1. 3-variable function 진리표

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

→ 다음은 3-variable function F(A,B,C)

에 관한 진리표이다.

→ 진리표는 임의로 수를 정하였다.

2. Function 구성 - Megawizard Plug-in manager를 이용하여 구현한 ROM

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   ENTITY lab_04 is
6      PORT (clk      : IN std_logic;
7            s        : IN std_logic_vector(2 DOWNTO 0) ;
8            F        : OUT std_logic_vector(2 DOWNTO 0) );
9   END lab_04 ;
10
11  ARCHITECTURE sample of lab_04 is
12
13  component rom_example
14     PORT ( address       : IN std_logic_vector(2 DOWNTO 0);
15            Clock         : IN std_logic ;
16            q             : OUT std_logic_vector(2 DOWNTO 0));
17  END component;
18
19  BEGIN
20     rom_example_inst:rom_example PORT MAP(
21           address  => s,
22           Clock    => clk,
23           q        => F );
24
25  END sample ;
```

→ Megawizard Plug-in manager를 이용하여 ROM을 VHDL Code를 이용하여 구현해보았다.

→ 강의자료를 토대로 ROM을 만들고 (rom_example Component를 만듦), 이를 이용하여 앞서 작성한 진리표의 결과가 나오도록 프로그래밍 하였다.

→ 해당 Logic은 clock과 관계없는 Combinational Logic인지만 실습 Tool인 Cyclone 특성상 Asynchronous Rom이 없기 때문에 clock이 포함되었다.

2. Function 구성 - mif 파일

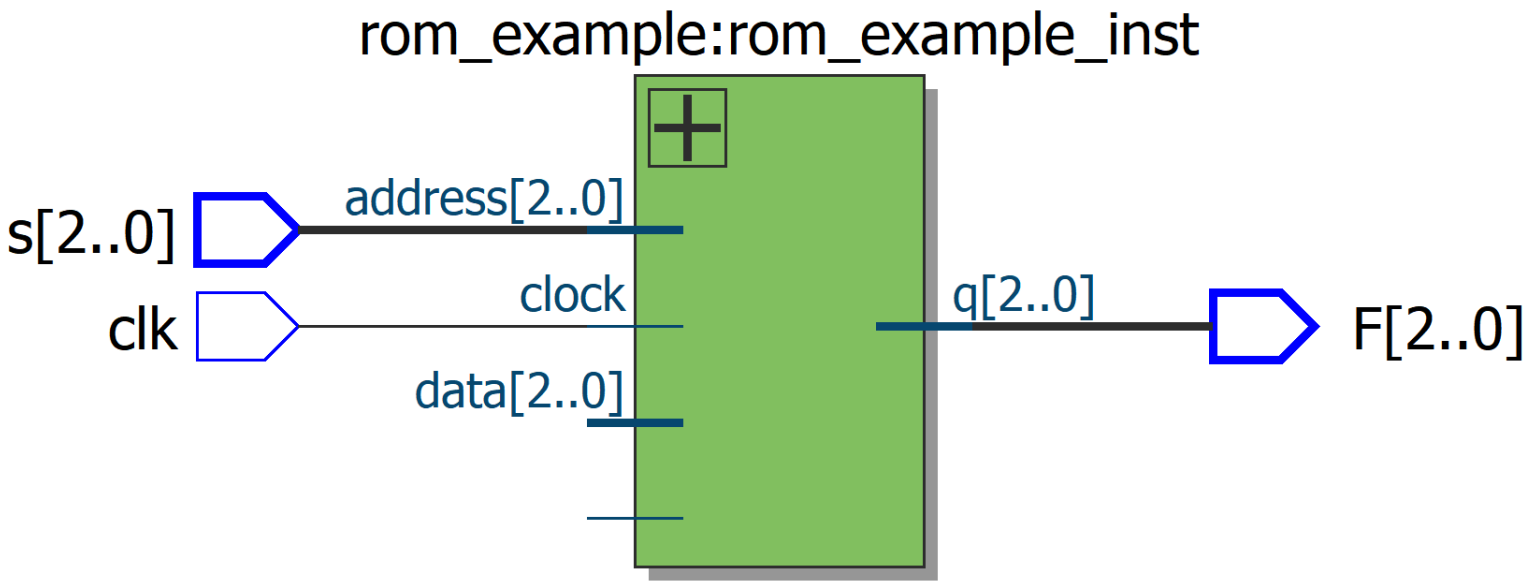
```
depth = 8;
width = 3;
address_radix = bin;
data_radix = bin;

content
begin
    000 : 000;
    001 : 000;
    010 : 001;
    011 : 001;
    100 : 001;
    101 : 000;
    110 : 001;
    111 : 001;
end;
```

→ 3-variable function이므로 Array 당 word의 width는 3 bit이고 Array의 개수인 depth는 8이다. 즉, 3 bit word가 8개 존재한다.

→ address_radix와 data_radix는 모두 이진수로 나타내므로 Binary(Bin)이다.

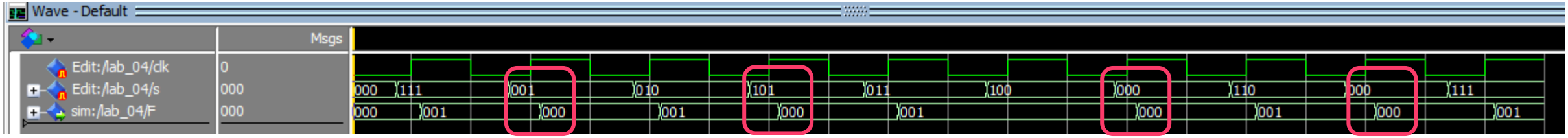
→ 해당하는 이진수는 진리표를 토대로 작성하였다.

2. Function 구성 – RLT Viewer



→ 앞선 VHDL Code를 RTL Viewer 를 통해 나타내면 다음과 같다.

3. Simulation capture



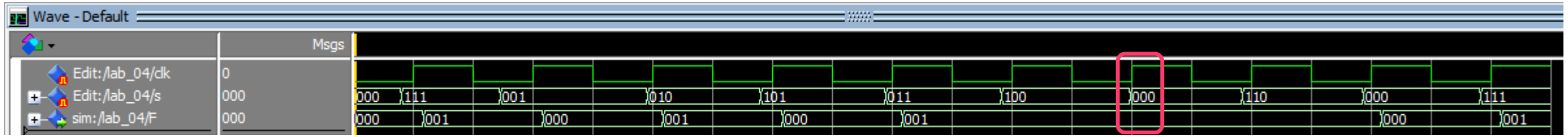
A	B	C	F	A	B	C	F
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	1

→ 해당 Simulation과 진리표에서 F가 0인 부분만 표시하였다.

→ 해당 Simulation은 임의로 숫자를 수정하여 결과를 확인하였다.

→ 왼쪽에 있는 진리표와 비교하여 결과를 확인해보면 동일하게 나왔다는 것을 알 수 있다.

3. Simulation capture

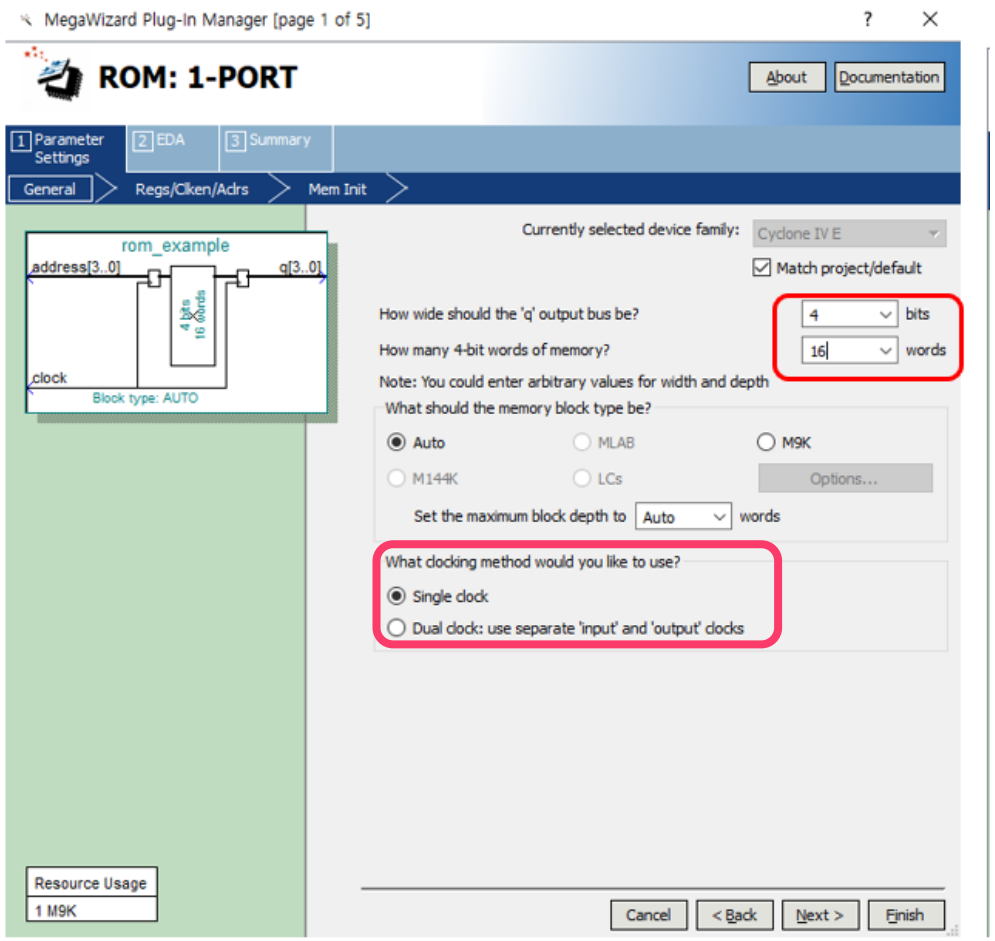


A	B	C	F	A	B	C	F
0	0	0	0	1	0	0	1
0	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1
0	1	1	1	1	1	1	1

→ 000부분에서 Input값을 받아들이지 못하고
약간의 Delay가 일어나며 값을 반영하지 못하였다.

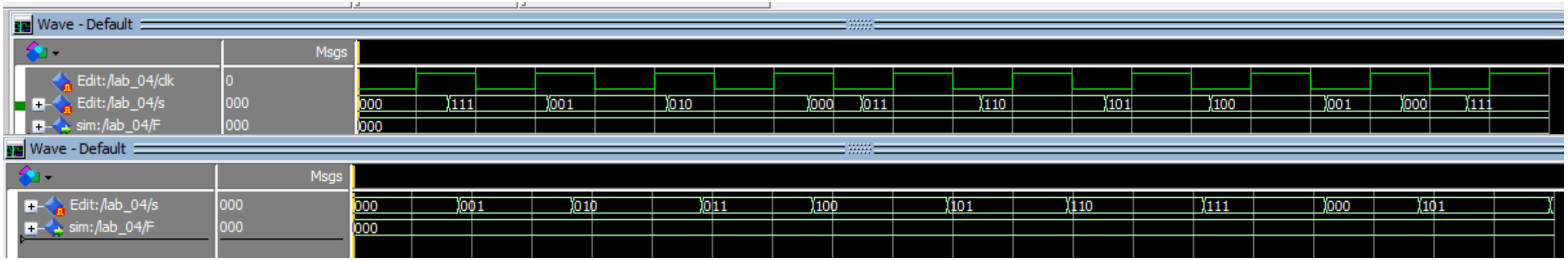
→ 해당 Simulation은 앞에 Simulation과 동일한데, 빨간 네모 부분은 Clock의 rising edge 근처에서 Data의 불안정으로 인하여 해당 Input이 반영되지 않았음을 확인할 수 있다.

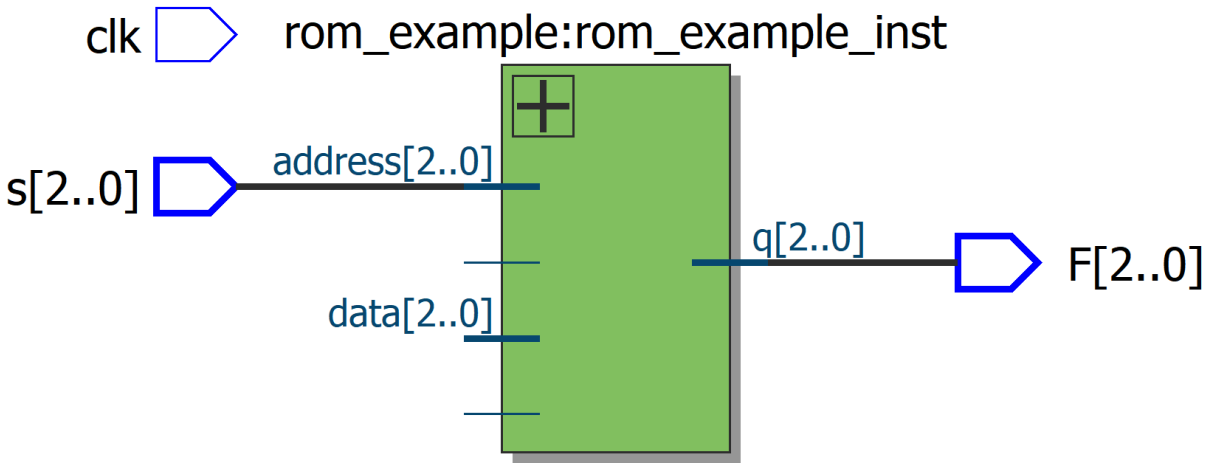
4. Discussion



→ 이번 Lab은 3-variable function에 대한 실습이라 Clock과 관계없는 Combinational Logic에 관한 실습이었다. 하지만 왼쪽과 같이 실습 Tool인 Cyclone의 특성상 Asynchronous Rom이 없으므로(clocking 방법을 무조건 선택하게 되어있다.) 해당 실습을 Clock 없이는 진행할 수 없어 결국 Clock을 포함하게 되었다.

4. Discussion





→ 해당 Simulation은 Clock을 연결하지 않거나 없애고 진행한 것인다 다음과 같이 결과가 전혀 나오지 않는 것을 확인할 수 있다. 이로 인해 부득이하게 Clock을 넣어 진행하였고, Simulation 결과가 Clock의 영향을 받을 수 밖에 없었다.

4. Discussion

→ 처음에는 Clock과 관계가 없는 Asynchronous Logic이므로 Clock의 존재를 완전히 없애기 위해 다양한 시도를 하였는데, Cyclone의 특성상 Clock을 없애지 못하였다.

→ 이번 실습을 통해 Asynchronous Logic과 Synchronous Logic의 차이를 자세히 알 수 있었고, Cyclone Tool의 특성에 대해서 잘 알 수 있었다.

→ 또한, 새로운 실습 방법인 Megawizard Plug-in manager를 이용하여 ROM을 구현한 것도 신기한 경험이었다.