Lab #9 RegFile

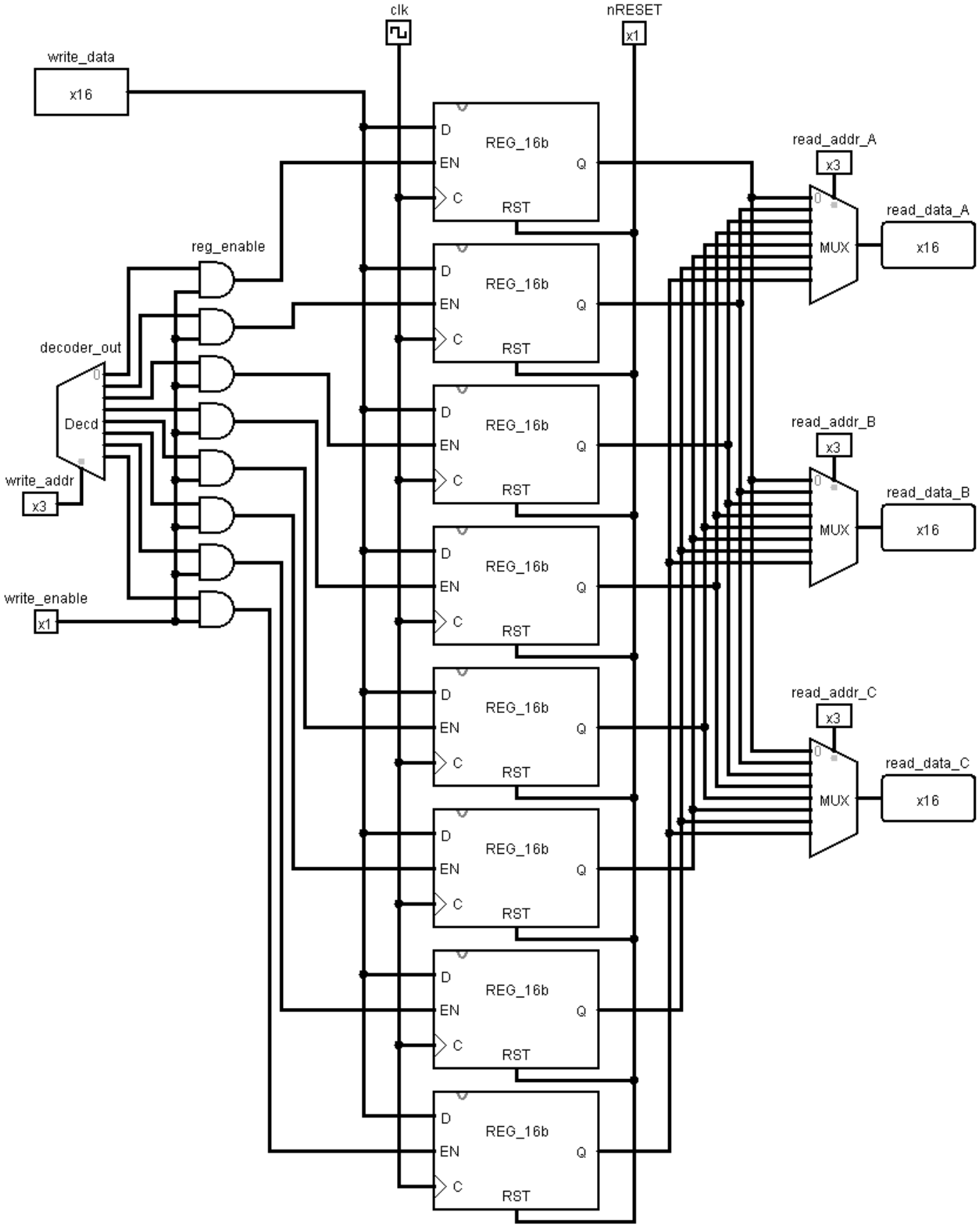
Class : 00

201602004 박태현

I. 실습 목적

8x16b 레지스터 구현

II. Design Procedure



III. Simulation

```
module register_file (clk, nRESET, write_enable, write_addr, write_data,
                      read_addr_A, read_addr_B, read_addr_C,
                      read_data_A, read_data_B, read_data_C);
    input clk;
    input nRESET;
    input write_enable;
    input [2:0] write_addr;
    input [15:0] write_data;

    // register index
    input [2:0] read_addr_A;
    input [2:0] read_addr_B;
    input [2:0] read_addr_C;

    // data in register
    output [15:0] read_data_A;
    output [15:0] read_data_B;
    output [15:0] read_data_C;

    // register
    reg [15:0] reg_0;
    reg [15:0] reg_1;
    reg [15:0] reg_2;
    reg [15:0] reg_3;
    reg [15:0] reg_4;
    reg [15:0] reg_5;
    reg [15:0] reg_6;
    reg [15:0] reg_7;

    // muxing
    wire [7:0] decoder_out;
    wire [7:0] reg_enable;
```

입력을 연결하고 레지스터 번호를 0~7번을 할당하므로 3비트를 레지스터 번호로 받습니다. 출력은 16비트로 하고, 레지스터도 8개, 16비트로 구성했습니다.

입력 3비트를 디코딩하여 8비트로 만들고, 이를 쓰기연산과 비교해 레지스터에 쓸지 결정하도록 했습니다

```
assign decoder_out =
    (write_addr == 3'b000) ? 8'b0000_0001 :
    (write_addr == 3'b001) ? 8'b0000_0010 :
    (write_addr == 3'b010) ? 8'b0000_0100 :
    (write_addr == 3'b011) ? 8'b0000_1000 :
    (write_addr == 3'b100) ? 8'b0001_0000 :
    (write_addr == 3'b101) ? 8'b0010_0000 :
    (write_addr == 3'b110) ? 8'b0100_0000 :
    (write_addr == 3'b111) ? 8'b1000_0000 : 8'bx;
    |
assign reg_enable[0] = write_enable & decoder_out[0];
assign reg_enable[1] = write_enable & decoder_out[1];
assign reg_enable[2] = write_enable & decoder_out[2];
assign reg_enable[3] = write_enable & decoder_out[3];
assign reg_enable[4] = write_enable & decoder_out[4];
assign reg_enable[5] = write_enable & decoder_out[5];
assign reg_enable[6] = write_enable & decoder_out[6];
assign reg_enable[7] = write_enable & decoder_out[7];
```

우선 쓰기를 할 주소를 쓰기할 레지스터 번호로부터 디코딩하여 위치를 찾습니다. 그리고, 쓰기가 허용되었는지 검사를 해서 그 레지스터에 쓸지를 연산하여 저장합니다.

```
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_0 <= 16'b0;
   else if (reg_enable[0]) reg_0 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_1 <= 16'b0;
   else if (reg_enable[1]) reg_1 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_2 <= 16'b0;
   else if (reg_enable[2]) reg_2 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_3 <= 16'b0;
   else if (reg_enable[3]) reg_3 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_4 <= 16'b0;
   else if (reg_enable[4]) reg_4 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_5 <= 16'b0;
   else if (reg_enable[5]) reg_5 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_6 <= 16'b0;
   else if (reg_enable[6]) reg_6 <= write_data;
always @ (posedge clk or negedge nRESET)
   if (!nRESET) reg_7 <= 16'b0;
   else if (reg_enable[7]) reg_7 <= write_data;
```

만약 리셋이 0이라면 각 레지스터를 모두 0으로 초기화를 합니다. 리셋이 아니라면 해당 레지스터에 쓰기를 시행하는지 여부를 확인해서 쓰기를 실행하는 조건이라면 write_data로 들어온 정보를 레지스터에 새로 씁니다

```
assign read_data_A =
    (read_addr_A == 3'b000) ? reg_0 :
    (read_addr_A == 3'b001) ? reg_1 :
    (read_addr_A == 3'b010) ? reg_2 :
    (read_addr_A == 3'b011) ? reg_3 :

    (read_addr_A == 3'b100) ? reg_4 :
    (read_addr_A == 3'b101) ? reg_5 :
    (read_addr_A == 3'b110) ? reg_6 :
    (read_addr_A == 3'b111) ? reg_7 : 16'bx;

assign read_data_B =
    (read_addr_B == 3'b000) ? reg_0 :
    (read_addr_B == 3'b001) ? reg_1 :
    (read_addr_B == 3'b010) ? reg_2 :
    (read_addr_B == 3'b011) ? reg_3 :

    (read_addr_B == 3'b100) ? reg_4 :
    (read_addr_B == 3'b101) ? reg_5 :
    (read_addr_B == 3'b110) ? reg_6 :
    (read_addr_B == 3'b111) ? reg_7 : 16'bx;

assign read_data_C =
    (read_addr_C == 3'b000) ? reg_0 :
    (read_addr_C == 3'b001) ? reg_1 :
    (read_addr_C == 3'b010) ? reg_2 :
    (read_addr_C == 3'b011) ? reg_3 :

    (read_addr_C == 3'b100) ? reg_4 :
    (read_addr_C == 3'b101) ? reg_5 :
    (read_addr_C == 3'b110) ? reg_6 :
    (read_addr_C == 3'b111) ? reg_7 : 16'bx;
```

레지스터는 읽기 속도가 매우 빠르기에, 읽기는 그냥 읽어서 나중에 제어를 통해서 사용을 안하도록 합니다

A,B,C로 들어온 레지스터 번호를 읽어서 해당 레지스터에 저장된 정보를 돌려줍니다

- Testbench

```
`timescale 1ns / 10ps

module register_file_tb;

reg clk;
reg nRESET;
reg write_enable;
reg [2:0] write_addr;
reg [15:0] write_data;

reg [2:0] read_addr_A;
reg [2:0] read_addr_B;
reg [2:0] read_addr_C;

wire [15:0] read_data_A;
wire [15:0] read_data_B;
wire [15:0] read_data_C;

always #2.5 clk=~clk;

register_file register_file_i(
    .clk(clk),
    .nRESET(nRESET),
    .write_enable(write_enable),
    .write_addr(write_addr),
    .write_data(write_data),
    .read_addr_A(read_addr_A),
    .read_addr_B(read_addr_B),
    .read_addr_C(read_addr_C),
    .read_data_A(read_data_A),
    .read_data_B(read_data_B),
    .read_data_C(read_data_C)
);
```

레지스터 입력에 필요한 레지스터 입력과 읽어온 정보를 담아둘 wire를 선언하고 모듈에 연결합니다

```
initial begin

    $dumpvars;
    clk=1'b1;
    nRESET = 1'b0;
    write_enable=1'b0;
    write_addr=3'b0;
    write_data=16'b0;
    read_addr_A=3'b0;
    read_addr_B=3'b0;
    read_addr_C=3'b0;
    #5
```

시작하면 초기화를 해줍니다

```
// 저장안됨
write_enable=1'b1;
write_addr=3'b000;
write_data=16'b1111_1111_0000_0000;
#5

write_enable=1'b1;
write_addr=3'b010;
write_data=16'b0000_0000_1111_1111;
#5
```

nRESET=0이므로 리셋 상태입니다. 이 상태에서 쓴 정보는 저장되지 않습니다

```
// 저장됨
nRESET = 1'b1;
write_enable=1'b1;
write_addr=3'b110;
write_data=16'b0000_0001_0000_0000;
#5

write_enable=1'b1;
write_addr=3'b111;
write_data=16'b0000_0000_0000_0001;
#5

write_enable=1'b1;
write_addr=3'b100;
write_data=16'b0000_0000_1000_0000;
#5
```

리셋을 1로 바꾸고 저장을 한 것은 저장이 됩니다

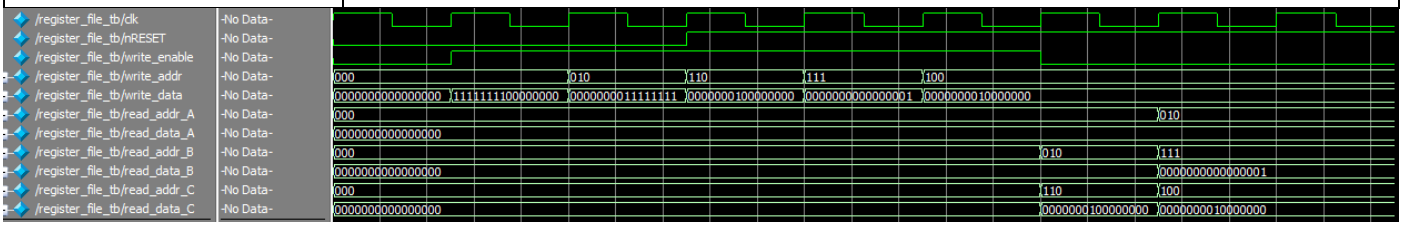
```
write_enable=1'b0;
read_addr_A=3'b000;
read_addr_B=3'b010;
read_addr_C=3'b110;
#5

write_enable=1'b0;
read_addr_A=3'b010;
read_addr_B=3'b111;
read_addr_C=3'b100;

#10;
$dumpoff;
$finish;
```

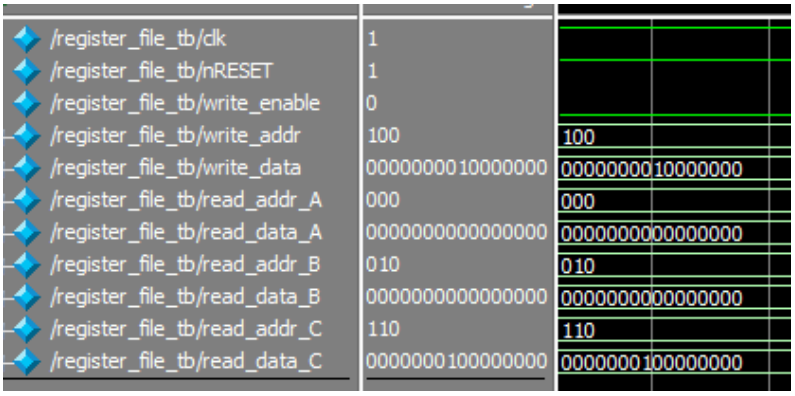
쓰기를 중단하여 더 쓰지 않도록 만들고, 값을 읽어옵니다

IV. Evaluation

쓰기를 수행한 후 레지스터의 값은 아래와 같이 설정됩니다

0	16'b0
1	16'b0
2	16'b0
3	16'b0
4	16'b0000_0000_1000_0000
5	16'b0
6	16'b0000_0001_0000_0000
7	16'b0000_0000_0000_0001

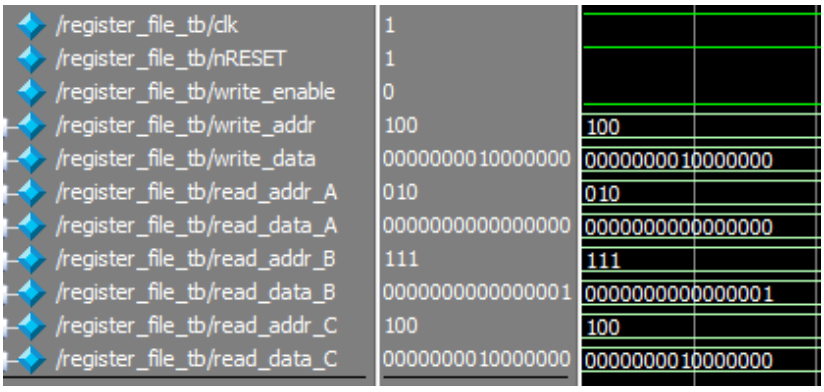




000 = 0

010 = 0

110 = 0000_0001_0000_0000



010 = 0

111 = 0000_0000_0000_0001

100 = 0000_0000_1000_0000

로 제대로 저장이 되고 읽어온 것을 볼 수 있었습니다

V. Discussion

복잡해보였던 레지스터를 베릴로그에서 간단히 구현할 수 있다는 것이 신기했습니다