SoC설계

Lab#11

Register File

컴퓨터공학과

201402439

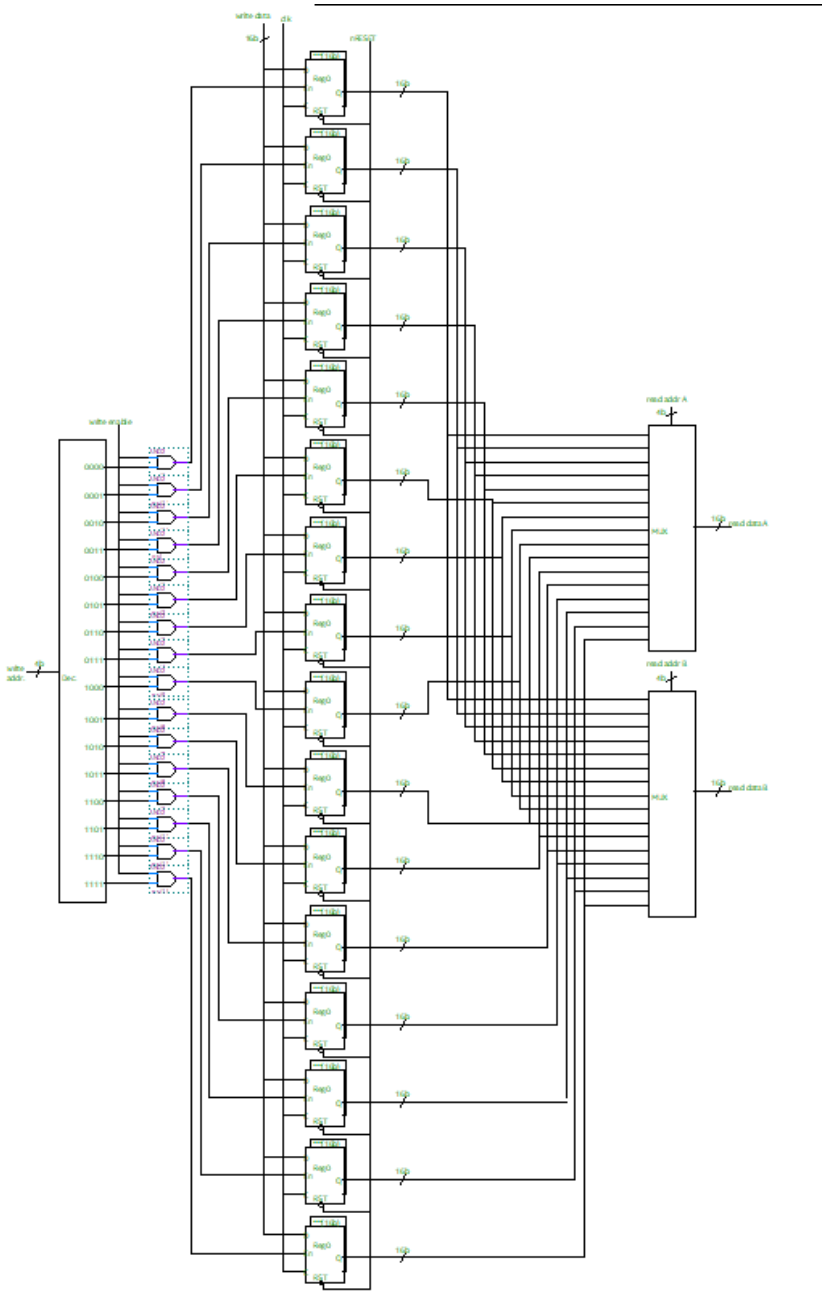
천원준

1. Purpose of the lab

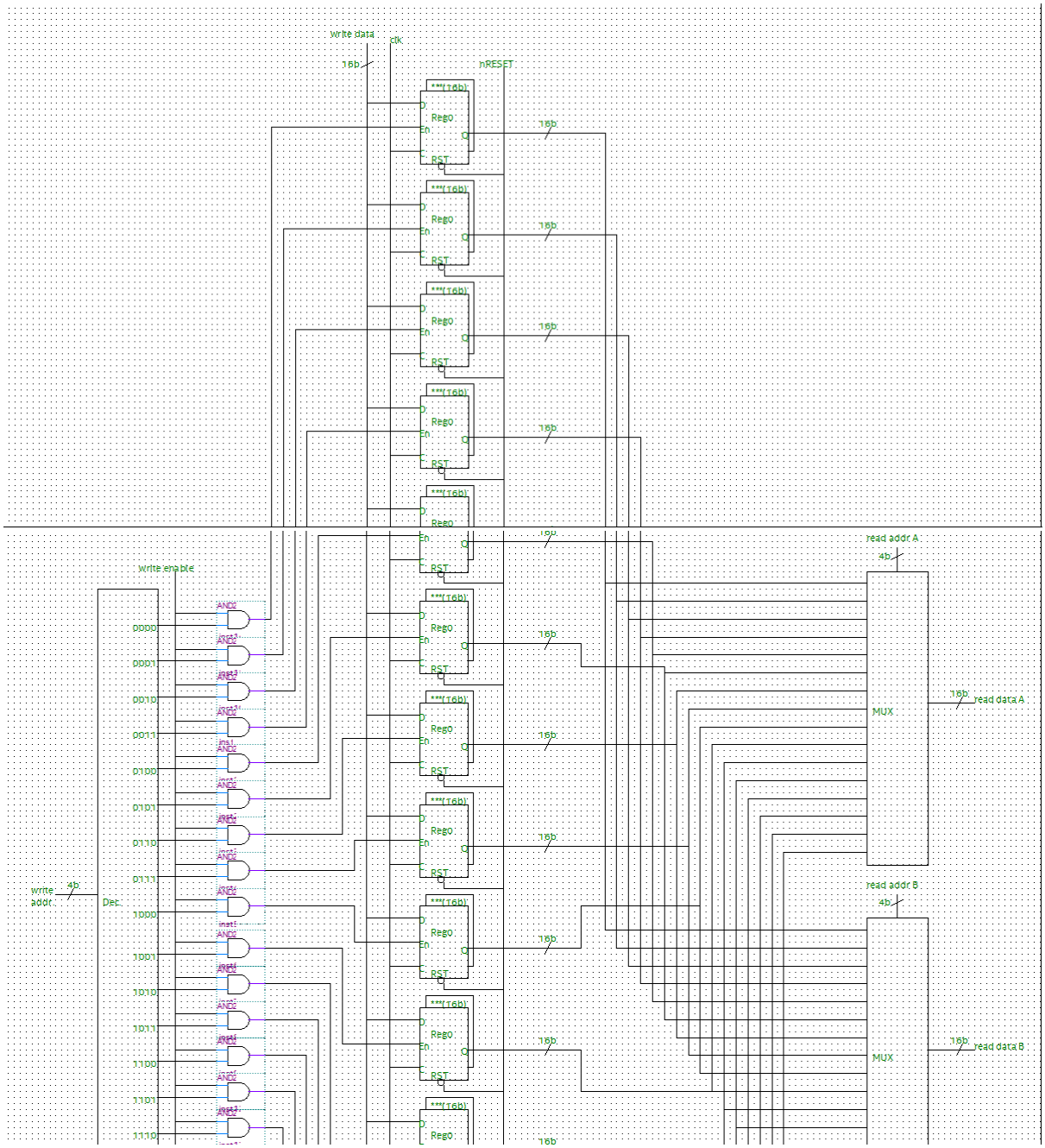
이번 실습의 목표는 4bit 주소와 16bit 데이터로 구성된 하나의 입력을 받아 해당 주소에 데이터를 입력하고, 4bit 주소와 16bit 데이터로 구성된 두개의 값을 읽어와 출력하는 2R1W 16*16b Register File를 구현하는 것입니다.

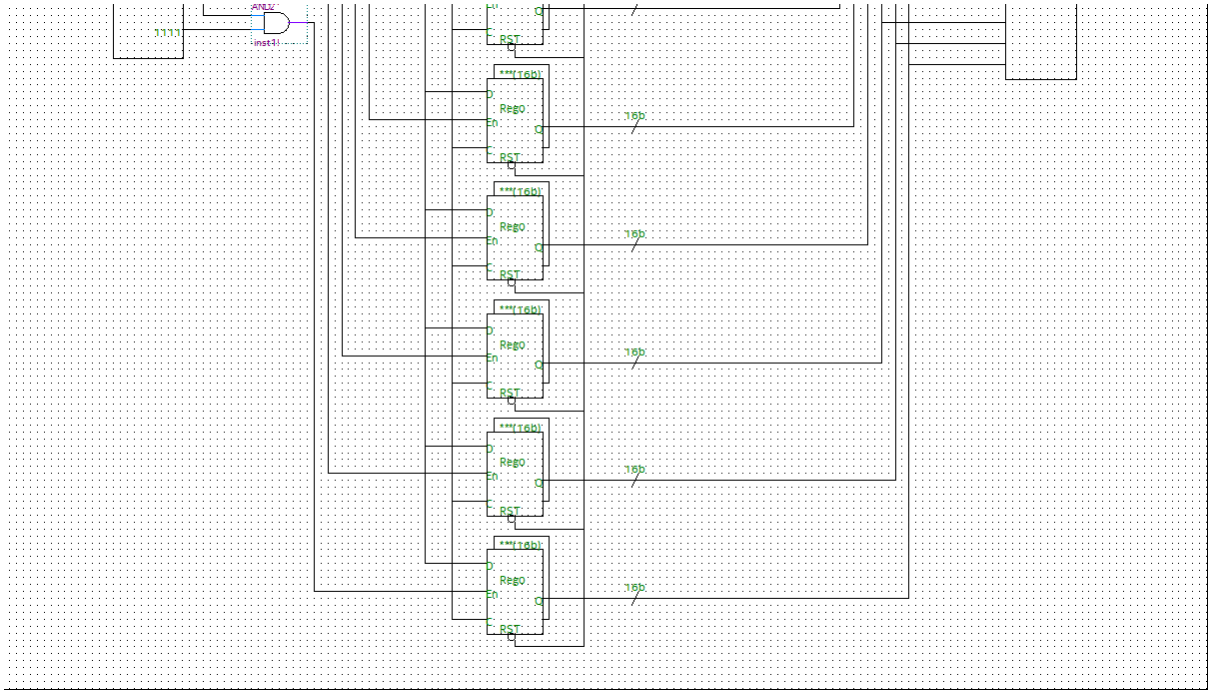
2. Design Procedure

- 전체 Block Diagram



- 확대한 block diagram





3. Simulation

- RegFile

```
module RegFile(clk, nRESET, write_en, write_addr, write_data, read_addr_A, read_addr_B, read_data_A, read_data_B);

    input clk, nRESET, write_en;
    input [3:0] write_addr, read_addr_A, read_addr_B;
    input [15:0] write_data;

    output [15:0] read_data_A, read_data_B;

    //register
    reg [15:0] reg_0, reg_1, reg_2, reg_3, reg_4, reg_5,
       reg_6, reg_7, reg_8, reg_9, reg_10, reg_11, reg_12,
       reg_13, reg_14, reg_15;

    //decoder output
    wire [15:0] decode_out;
    wire [15:0] reg_en;

    assign decode_out =
        (write_addr == 4'b0000) ? 16'b0000000000000001:
        (write_addr == 4'b0001) ? 16'b0000000000000010:
        (write_addr == 4'b0010) ? 16'b0000000000000100:
        (write_addr == 4'b0011) ? 16'b0000000000001000:
        (write_addr == 4'b0100) ? 16'b0000000000010000:
        (write_addr == 4'b0101) ? 16'b0000000000100000:
        (write_addr == 4'b0110) ? 16'b0000000001000000:
        (write_addr == 4'b0111) ? 16'b0000000010000000:
        (write_addr == 4'b1000) ? 16'b0000000100000000:
        (write_addr == 4'b1001) ? 16'b0000001000000000:
        (write_addr == 4'b1010) ? 16'b0000010000000000:
        (write_addr == 4'b1011) ? 16'b0000100000000000:
        (write_addr == 4'b1100) ? 16'b0001000000000000:
        (write_addr == 4'b1101) ? 16'b0010000000000000:
        (write_addr == 4'b1110) ? 16'b0100000000000000:
        (write_addr == 4'b1111) ? 16'b1000000000000000:
       16'bx;
```

우선, clk, nRESET, 쓸 것인지 결정하는 write_en, 쓸 주소 write_addr, 쓸 데이터 write_data, 읽을 주소 read_addr_A, read_addr_B를 입력으로 받고, 읽은 값 read_data_A, read_data_B를 출력으로 갖습니다.

그리고 4비트의 주소값으로 16개의 플립플롭 묶음에 찾아가 쓸 수 있도록 decoding해주는 decode 신호를 decode_out에 할당해 줍니다.

```
assign reg_en[0] = write_en & decode_out[0];
assign reg_en[1] = write_en & decode_out[1];
assign reg_en[2] = write_en & decode_out[2];
assign reg_en[3] = write_en & decode_out[3];
assign reg_en[4] = write_en & decode_out[4];
assign reg_en[5] = write_en & decode_out[5];
assign reg_en[6] = write_en & decode_out[6];
assign reg_en[7] = write_en & decode_out[7];
assign reg_en[8] = write_en & decode_out[8];
assign reg_en[9] = write_en & decode_out[9];
assign reg_en[10] = write_en & decode_out[10];
assign reg_en[11] = write_en & decode_out[11];
assign reg_en[12] = write_en & decode_out[12];
assign reg_en[13] = write_en & decode_out[13];
assign reg_en[14] = write_en & decode_out[14];
assign reg_en[15] = write_en & decode_out[15];
```

그리고 각 플립플롭 묶음에 enable 신호와 decode 신호가 들어왔을 때, 플립플롭에 값을 쓸 수 있도록 reg_en값을 할당해 줍니다.

```
always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_0 <= 16'b0;
   else if(reg_en[0])
      reg_0 <= write_data;
end

always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_1 <= 16'b0;
   else if(reg_en[1])
      reg_1 <= write_data;
end

always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_2 <= 16'b0;
   else if(reg_en[2])
      reg_2 <= write_data;
end

always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_3 <= 16'b0;
   else if(reg_en[3])
      reg_3 <= write_data;
end

always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_4 <= 16'b0;
   else if(reg_en[4])
      reg_4 <= write_data;
end

always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_5 <= 16'b0;
   else if(reg_en[5])
      reg_5 <= write_data;
end
```

이제 각 플립플롭 묶음에 값을 써주는 단계입니다. Clk이 상승 모서리거나 리셋 신호가 활성화되면 플립플롭 묶음의 값이 갱신됩니다. 리셋 신호가 활성화되었을 경우에는 해당 주소의 플립플롭 묶음의 값을 0으로 초기화해줍니다. reg_en 신호가 활성화되었을 경우에는 write_data로 입력받은 16비트 값을 플립플롭 묶음에 써줍니다.

```
always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_14 <= 16'b0;
   else if(reg_en[14])
      reg_14 <= write_data;
end

always @ (posedge clk or negedge nRESET) begin
   if(!nRESET)
      reg_15 <= 16'b0;
   else if(reg_en[15])
      reg_15 <= write_data;
end
```

각 주소의 플립플롭 묶음에 값을 써주는 과정은 같으므로 중간은 생략합니다.

```
assign read_data_A =
   (read_addr_A == 4'b0000) ? reg_0 :
   (read_addr_A == 4'b0001) ? reg_1 :
   (read_addr_A == 4'b0010) ? reg_2 :
   (read_addr_A == 4'b0011) ? reg_3 :
   (read_addr_A == 4'b0100) ? reg_4 :
   (read_addr_A == 4'b0101) ? reg_5 :
   (read_addr_A == 4'b0110) ? reg_6 :
   (read_addr_A == 4'b0111) ? reg_7 :
   (read_addr_A == 4'b1000) ? reg_8 :
   (read_addr_A == 4'b1001) ? reg_9 :
   (read_addr_A == 4'b1010) ? reg_10 :
   (read_addr_A == 4'b1011) ? reg_11 :
   (read_addr_A == 4'b1100) ? reg_12 :
   (read_addr_A == 4'b1101) ? reg_13 :
   (read_addr_A == 4'b1110) ? reg_14 :
   (read_addr_A == 4'b1111) ? reg_15 : 16'bx;

assign read_data_B =
   (read_addr_B == 4'b0000) ? reg_0 :
   (read_addr_B == 4'b0001) ? reg_1 :
   (read_addr_B == 4'b0010) ? reg_2 :
   (read_addr_B == 4'b0011) ? reg_3 :
   (read_addr_B == 4'b0100) ? reg_4 :
   (read_addr_B == 4'b0101) ? reg_5 :
   (read_addr_B == 4'b0110) ? reg_6 :
   (read_addr_B == 4'b0111) ? reg_7 :
   (read_addr_B == 4'b1000) ? reg_8 :
   (read_addr_B == 4'b1001) ? reg_9 :
   (read_addr_B == 4'b1010) ? reg_10 :
   (read_addr_B == 4'b1011) ? reg_11 :
   (read_addr_B == 4'b1100) ? reg_12 :
   (read_addr_B == 4'b1101) ? reg_13 :
   (read_addr_B == 4'b1110) ? reg_14 :
   (read_addr_B == 4'b1111) ? reg_15 : 16'bx;
endmodule
```

마지막으로 레지스터에 저장된 값을 불러오는 단계입니다. 입력받은 read_addr를 이용하여 해당 주소의 플립플롭 묶음에 저장되어 있는 값을 read_data에 할당해줍니다.

- RegFile testbench

```
`timescale 1ns/100ps

module RegFile_tb;

   reg clk, nRESET, write_en;
   reg [3:0] write_addr, read_addr_A, read_addr_B;
   reg [15:0] write_data;

   wire [15:0] read_data_A, read_data_B;

   RegFile regfile(.clk(clk), .nRESET(nRESET),
                   .write_en(write_en), .write_addr(write_addr), .write_data(write_data),
                   .read_addr_A(read_addr_A), .read_addr_B(read_addr_B),
                   .read_data_A(read_data_A), .read_data_B(read_data_B));

   initial begin
      clk = 1;
      forever #5 clk = ~clk;
   end

   initial begin
      nRESET = 1;
      #5;
      nRESET = 0;
      #5;
      nRESET = 1;

      #5
      write_en = 1;
      write_addr = 4'b0000;
      write_data = 16'b1001011000011100;

      #15
      write_addr = 4'b0001;
      write_data = 16'b1111111111111111;

      #10
      write_en = 0;
      read_addr_A = 4'b0000;
      read_addr_B = 4'b0001;

      #10
      $finish;

   end
endmodule
```

우선 #5의 간격으로 clk 신호가 바뀌도록 만들었습니다. 그리고 0번 주소와 1번 주소에 값을 할당하였습니다. 마지막으로 0번 주소와 1번 주소의 값을 읽어오도록 한 후 종료하였습니다.

결과가 정상적으로 나오는 것을 확인하였습니다.

4. Evaluation

실행한 결과가 의도한 대로 나오는 것을 보아, 올바로 설계했음을 확인할 수 있습니다.

5. Discussion

이번 실습의 key는 각 플립플롭 묶음(레지스터)마다 주소가 있고, 해당 주소의 플립플롭 묶음에 값을 저장할 수 있다는 점입니다.

흥미로웠던 점은 두 값을 순차적으로 입력시킬 때 #10 이상의 시간간격을 두고 입력시키지 않으면 정상적으로 값이 써지지 않는다는 점이었습니다.

이 design을 향상시킬 방법은 없습니다.