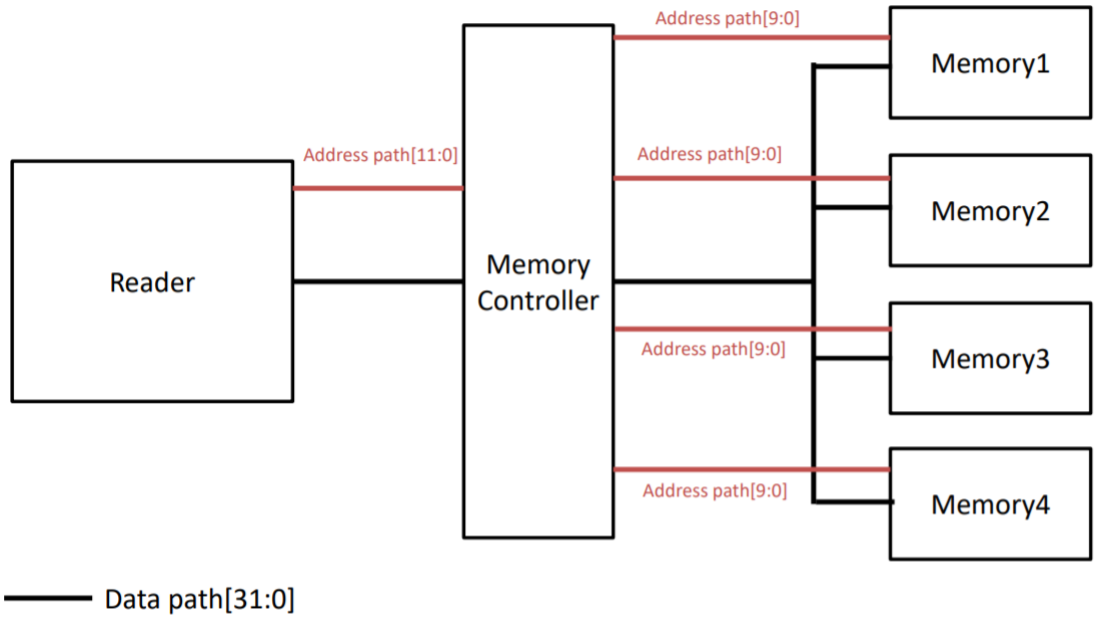
디지털시스템설계 LAB2

2016310936 우승민

LAB2 의 목표는 아래의 그림에서 **Memory Controller** 를 구현하는 것입니다. 데이터 전달과정을 요약하면 다음과 같습니다.



Memory Controller 가 **Reader** 에 allow_address 신호를 보내주면 **Reader** 에서 원하는 메모리주소 **12bit** 를 보내줍니다.

Memory address [11:0]

11	10	9	8	7	6	5	4	3	2	1	0

Memory Number[11:10] **Address in memory**

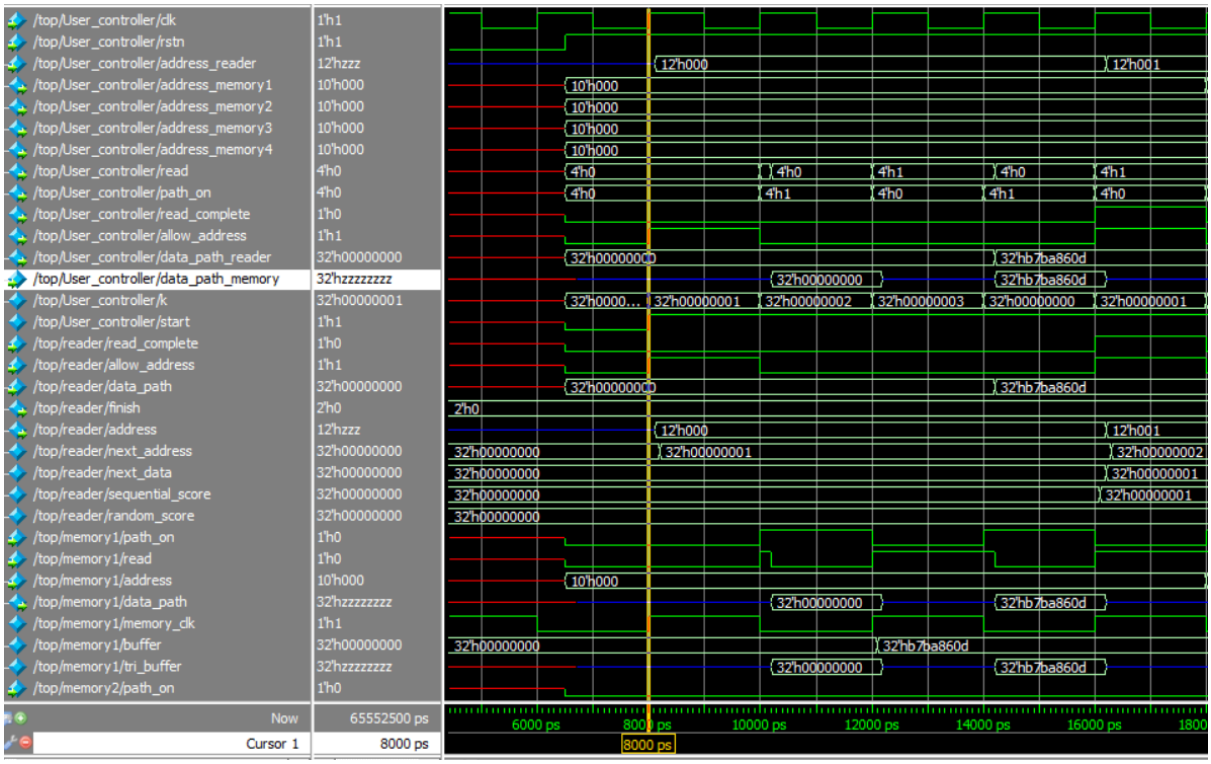
- 00:Memory1
- 01:Memory2
- 10:Memory3
- 11:Memory4

그 후 **Controller** 는 메모리주소의 상위 2 비트로 memory 구분 후 하위 10bit 를 보내줍니다. **Memory** 는 받은 주소에 해당하는 데이터를 읽어 Controller 에게 전달하고 Controller 는 받은 데이터를 **Reader** 에 전달합니다.

- **Memory1:** 4ns clock period, posedge active
- **Memory2:** 4ns clock period, negedge active
- **Memory3:** 8ns clock period, posedge active
- **Memory4:** 4ns clock period, negedge active, it gives back data 4ns later.

Code 설명에 앞서 말씀드리면, 저는 memory 마다 clock cycle 이 다르기 때문에 최대 cycle 인 8ns(8000ps)을 기준으로 code 를 만들어 주었습니다.

아래 timing graph 를 보면 rstn 이 1 로 설정된 **6500ps** 이후 첫 positive clock edge 인 **8000ps** 를 시작점으로 잡고 그 이후 **16000ps** 사이의 **8000ps** 동안 값들이 변화하는 것을 볼 수 있습니다.




```
integer i,k;
reg start;
always @ (posedge rstn) begin
        address_memory1 <= 0;
        address_memory2 <= 0;
        address_memory3 <= 0;
        address_memory4 <= 0;
        read <= 0;
        path_on <= 0;
        read_complete <= 0;
        allow_address <= 0;
        data_path_reader <= 0;
        k <= 0;
        start <= 0;
        for(i=0; i<BUFFER_SIZE; i = i+1) begin
                data_buffer[i] <= 0;
                address_buffer[i] <= 0;
        end
end
```

.

사용하는 값들은 rstn 이 변화하는 동시에 초기화해주었습니다

가장 처음 진행되는 **reader** 로부터 원하는 address 를 받는 과정을 설명하겠습니다.

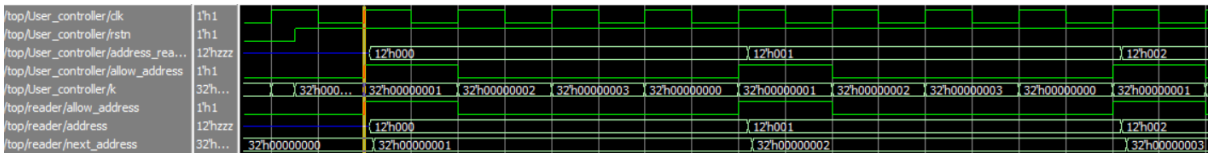
Controller

```
always @ (posedge clk) begin
        if(k ==0) begin
          allow_address <= 1;
          if(start !== 0) read_complete <= 1;
          else start <= 1;
        end
        if(k !== 32'bx) k <= k+1;
        if(k == 3) k <= 0;
        if(allow_address == 1) allow_address <=0;
```

Reader

```
if(allow_address == 1) begin //give address
        address <= read[next_address];
        #0.1 next_address <= next_address + 1;
end
```

저는 positive clock edge 일 때 마다 k 값을 3 까지 1 씩늘려 주었고, k 가 0 이 될 때마다 allow_adress 값에 1 을 넣어주었습니다. 정리하면 **4 clock cycle(8000ps)**마다 allow_address 값에 1 을 넣어준 것입니다.



Timing graph 를 보면 k 값이 1 이되는 동시에 allow_address 값이 1 이 되고 100ps 후에 address 값과 address_reader 값이 입력되는 것을 확인할 수 있습니다. 그 뒤 100ps 가 지나면 next_address 에 1 이 더해집니다.

Reader 부터 address 값을 받고 **memory** 로 전달되는 과정은 다음 코드에서 일어납니다.

Controller

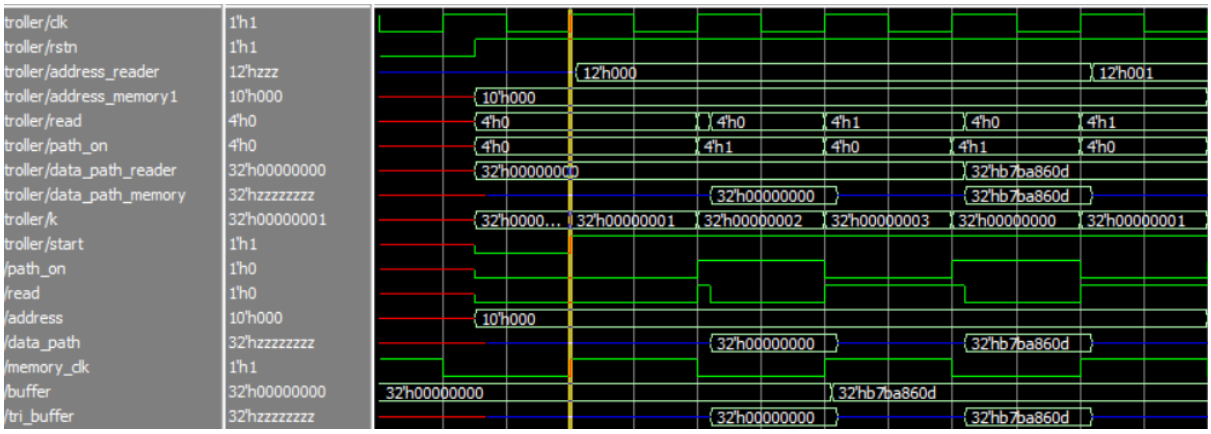
```
if(k == 0 || k == 2) path_on <=0;
case(address_reader[11:10])
2'b00 : begin
        if(k == 1 || k==3) path_on <= 4'b0001;
        read <= 4'b0001;
        address_memory1[9:0] <= address_reader[9:0];
        end
2'b01 : begin
        if(k == 1 || k==3) path_on <= 4'b0010;
        read <= 4'b0010;
        address_memory2[9:0] <= address_reader[9:0];
        end
2'b10 : begin
        if(k == 1 || k==3) path_on <= 4'b0100;
        read <= 4'b0100;
        address_memory3[9:0] <= address_reader[9:0];
        end
2'b11 : begin
        if(k == 1 || k==3) path_on <= 4'b1000;
        read <= 4'b1000;
        address_memory4[9:0] <= address_reader[9:0];
        end
endcase
```

memory1

```
assign data_path = tri_buffer;

always @ (*) begin
        #CLOCK_PERIOD memory_clk <= ~memory_clk;
end

always @ (path_on) begin
        if(path_on == 1'b1) begin
                #0.2 tri_buffer <= buffer;
        end
        else begin
                #0.2 tri_buffer <= 'bZ;
        end
end

//give data to data_path
always @ (posedge memory_clk) begin
        #0.1
        if(read == 1'b1) begin
                buffer <= data[address];
        end
end
```

case 문을 사용하여 address_reader 의 상위 2bit 로 구분하여 해당되는 memory 에 하위 10bit 값을 넣어주고 read 와 path_on 값도 해당되는 memory 에 맞게 설정해 줍니다. memory 에서는 read 값이 설정되면 buffer 에 해당 주소에 해당하는 data 값을 넣어주고 path_on 값이 설정되면 tri_buffer 즉 **data_path** 에 buffer 에 넣었던 data 값을 넣어줍니다.

path_on 값은 k 를 사용하여 2clock cycle 마다 1 로 설정해주었습니다 tri_buffer 에 data 를 넣는 것은 path_on 이 변화할 때 일어나기 때문입니다.



Timing graph 를 보면 positive memory_clock edge 의 **100ps** 이후에 buffer 에 data 가 입력됩니다. 동시에 path_on 이 1 -> 0 -> 1 로 변화한 후 **200ps** 후 tri_buffer 에 buffer 에 넣어준 data 값을 넣어 줍니다. Data_path 와 data_path_memory 는 tri_buffer 와 같도록 assign 되어 있기 때문에 동시에 변화합니다.

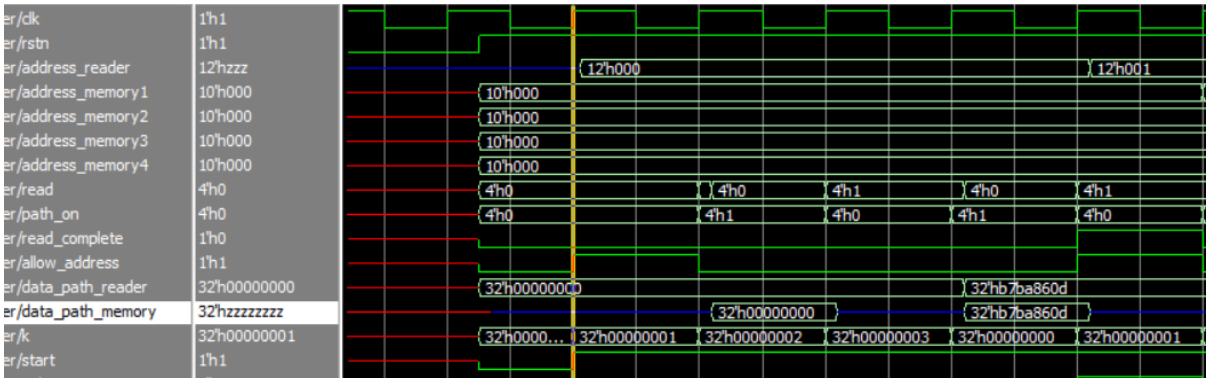
아래 code 는 **controller** 에서 **memory** 에게 받은 data 값을 reader 로 넘겨주는 과정입니다.

```
always @ (data_path_memory) begin
        if(data_path_memory !== 32'bx && data_path_memory !== 32'bz) begin
                data_path_reader <= data_path_memory;
                read <= 0;
        end
end
```

Data_path_memory 값이 변화하면 data_path_reader 에 값을 넣어주고, read 를 0 으로 초기화합니다.

```
always @ (posedge clk) begin
        if(k ==0) begin
          allow_address <= 1;
          if(start !== 0) read_complete <= 1;
          else start <= 1;
        end
        if(k !== 32'bx) k <= k+1;
        if(k == 3) k <= 0;
        if(allow_address == 1) allow_address <=0;
        if(read_complete ==1) read_complete <= 0;
```

read_complete 는 allow_address 와 달리 **start** 라는 변수가 1 인 상태여야 값을 넣게 해주었는데 이유는 timing graph 를 보시면 알 수 있습니다.

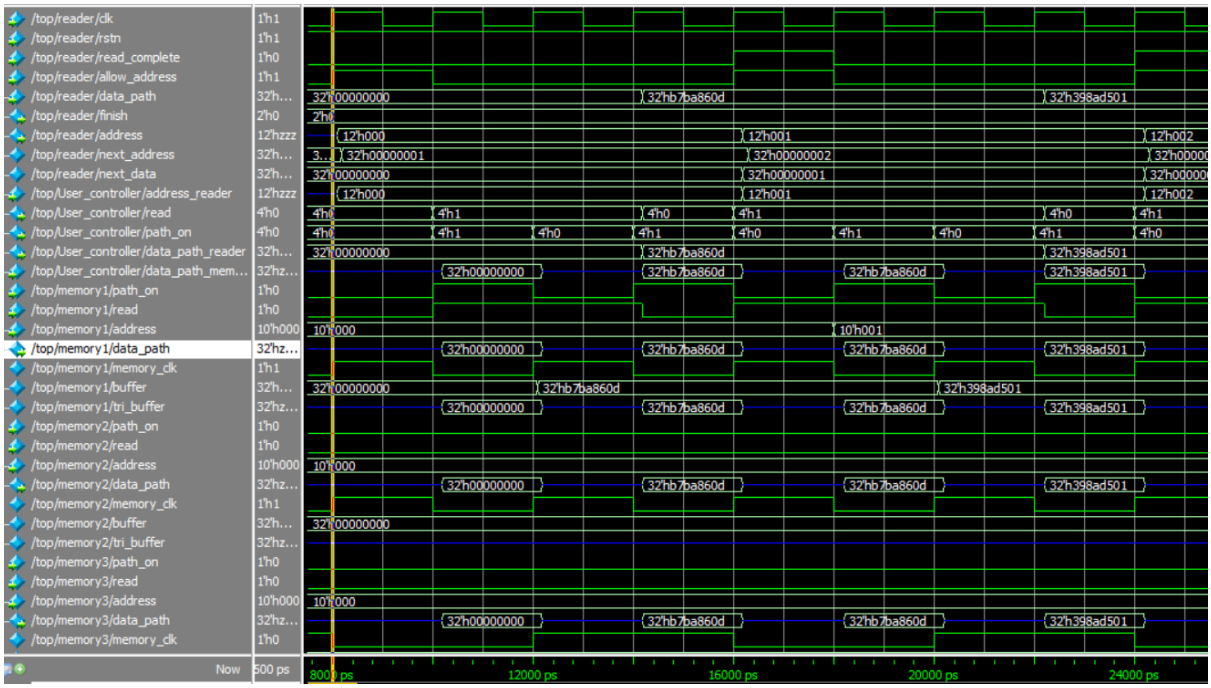


처음부터 allow_address 와 같이 read_complete 를 키게 되면 아직 data_path 에 값이 입력되지 않은 상태를 reader 에 넘겨줄 수 있기 때문에 start 라는 변수를 rstn 이 켜지고 첫 positive clock edge 인 **8000ps** 에서 켜준 이후 **4clock cycle(8000ps)**마다 allow_address 와 같이 키게 하였습니다.

reader 에서는 read_complete 값이 1 이 되면 data 와 정답을 확인하고 **100ps** 후에 next_data 값을 1 더해줍니다.

```
if(read_complete == 1 && finish == 2'b00) begin//sequential read
        if(data[next_data] == data_path)begin
                sequential_score <= sequential_score + 1;
                $display("answer : %b your data : %b <right>",data[next_data],data_path);
        end
        else begin
                $display("answer : %b your data : %b <wrong>",data[next_data],data_path);
        end
        #0.1 next_data <= next_data +1;
end
```

read_complete 가 1 이 되고 **100ps** 이후 reader 에서 전달받은 data 와 답을 확인하여 sequential_score 에 1 을 더해주고 **100ps** 추가로 지난 후 next_data 값을 1 늘려줍니다.



정리하면, allow_address 를 설정하면 **100ps** 에 reader 로부터 address 를 받고 그 다음 positive clock edge 에(**2000ps**) 해당되는 memory 에 주소와 함께 read, path_on 을 설정해줍니다.

여기서

1. memory 1 의 경우는 1 로 시작하고 4000ps 의 주기를 가지며 memory clock 이 positive 일때 read 가 동작하므로 **4100ps** 에 buffer 에 data 가 입력
   6000ps 에 path_on 이 켜지며 **6200ps** 에 data_path 에 data 입력

2. memory 2 의 경우는 1 로 시작하고 4000ps 의 주기를 가지며 memory clock 이 negative 일때 read 가 동작하므로 **2100ps** 에 buffer 에 data 가 입력
   6000ps 에 path_on 이 켜지며 **6200ps** 에 data_path 에 data 입력

3. memory 3 의 경우는 0 으로 시작하고 8000ps 의 주기를 가지며 memory clock 이 positive 일때 read 가 동작하므로 **4100ps** 에 buffer 에 data 가 입력
   6000ps 에 path_on 이 켜지며 **6200ps** 에 data_path 에 data 입력

4. memory 4 의 경우는 1 로 시작하고 4000ps 의 주기를 가지며 memory clock 이 negative 일때 read 가 동작하므로 **2100ps** 에 buffer1 에 data, **4100ps** 에 buffer 에 buffer1 이 입력
   6000ps 에 path_on 이 켜지며 **6200ps** 에 data_path 에 data 입력

clock 주기를 맞추어 모든 memory 가 **2000ps~6200ps** 사이에 controller 에 data 를 전달하게 해주었고 그 다음 positive clock edge 인 **8000ps** 에 read_complete 와 allow_address 를 동시에 켜주며 **8100ps** 에 정답 확인, **8200ps** 에 next_data 값 증가, **8300ps** 에 next_address 변경을 하여 위의 과정을 다시 할 수 있게 해주었습니다.

이렇게 **8000ps** 의 주기로 **reader** 와 **memory** 사이를 **control** 하였습니다.

아래는 code 에 test bench 를 실행하였을 때 결과입니다.

```
# answer : 10001101011101010000010000101101 your data : 10001101011101010000010000101101 <right>
# sequential read score :        4096/4096 random read score :        4096/4096
# ** Note: $finish    : C:/classes/digital/lab2/lab2_reader.v(76)
#    Time: 65552500 ps  Iteration: 0  Instance: /top/reader
# 1
# Break in Module memory_reader at C:/classes/digital/lab2/lab2_reader.v line 76
```

계산해보면 8000ps X 8192 = 65,536,000 ps 이므로 추가적인 delay 값(ex. rstn, finish)들을 더해지면 실제 값과 동일하게 나옵니다.