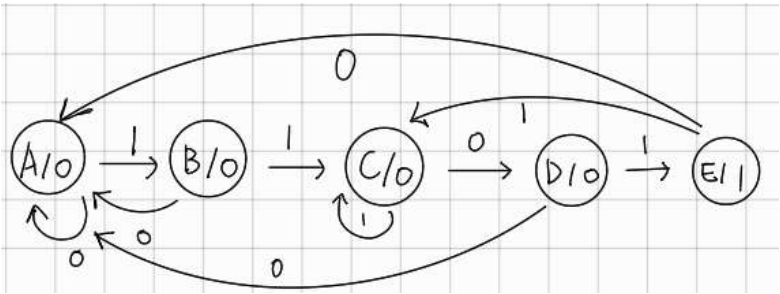
14주차 결과보고서

전공: 경영학과 학년: 4학년 학번: 20190808 이름: 방지혁

1.

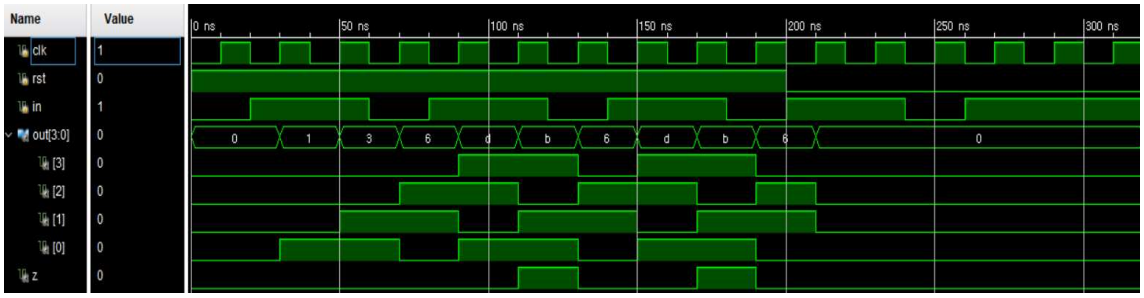
<Sequence Detector 1101 Moore Machine State Diagram>



<Sequence Detector 1101 Moore Machine State Table>

q (current state)	q* (next state)		z (output)
	x = 0	x = 1	
A	A	B	0
B	A	C	0
C	D	C	0
D	A	E	0
E	A	C	1

Sequence Detector 1101의 Moore machine 상태표에서는 A부터 E까지의 5가지 상태가 존재합니다. E 상태의 경우에만 output인 z가 1으로 출력됩니다. 나머지의 경우는 다 output이 0입니다. A의 상태에서 input이 1이면 B로 전이합니다. B의 상태에서 input이 1이면 C로 전이하고, 반면, C의 상태에서는 0이 input으로 들어가면 D로 전이하고, 마지막으로 D의 상태에서는 input이 1이면 E 상태로 전이하여 1101의 시퀀스를 감지하여 output으로 1이 출력됩니다.



보시는 바와 같이 시뮬레이션 결과가 나왔습니다. 우리가 예비 보고서에서 조사한 바와 같이 Moore machine은 state만을 고려합니다. 시뮬레이션 결과에서 알 수 있듯이 1101의 sequence가 차례로 들어왔을 때 바로 output으로 1이 출력되는 것이 아니라 다음 state, 즉 그 다음 clk의 postitive edge에서 출력이 되는 것을 확인할 수 있습니다. 그리고 보드의 rst 스위치가 Active Low이기 때문에 rst이 0일 때 정상 동작하도록 했습니다.

```
`timescale 1ns / 1ps

module moore(
input clk,rst, in,
output reg [3:0] out,
output reg z
);

initial begin
out = 4'b0;
z = 1'b0;
end

always@(posedge clk) begin
  if (rst == 0) begin
    out = 4'b0;
    z = 1'b0;
  end

  else begin
    if (out == 4'b1101)
      z = 1'b1;
    else
      z = 1'b0;
    out = out << 1;
    out[0] = in;
  end
end

endmodule
```

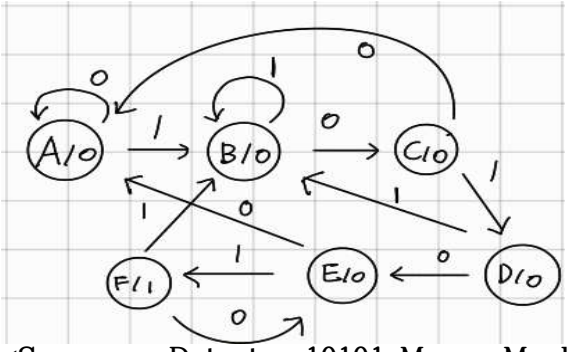
```
`timescale 1ns / 1ps

module moore_tb;
reg clk,rst, in;
wire [3:0] out;
wire z;

moore test(
.clk(clk), .rst(rst), .in(in), .out(out), .z(z)
);
initial begin
clk = 1'b0;
rst = 1'b1;
in = 1'b0;
#20 in = 1'b1;
#40 in = 1'b0;
#20 in = 1'b1;
#40 in = 1'b0;
#20 in = 1'b1;
#40 in = 1'b0;
#20 in = 1'b1;
#40 in = 1'b0;
#20 in = 1'b1;
#60
$finish;
end

always@(clk or rst) begin
rst <= #200 ~rst;
clk <= #10 ~clk;
end
endmodule
```

input으로는 clk, rst, in을 선언하고 output으로는 reg형으로 크기 4짜리 배열 [3:0] out과 시퀀스가 탐지되었을 때 1을 출력할 z를 선언해줍니다. 또한 시뮬레이션상 편의를 위해 초기 output 값을 모두 0으로 설정해줍니다. 4개의 out중 out[0]이 가장 최근에 들어온 bit임을 가정하고 코드를 작성합니다. 해당 모듈은 clk의 positive edge에서만 작동하고 rst으로 0이 들어올 경우 4개의 out과 z를 모두 0으로 초기화해줍니다. 그 외의 경우에는 좌측으로 1비트씩 shift 연산(<<)을 하고 가장 최근에 들어온 bit를 저장하는 out[0]에 in값을 넣어줍니다. 또한, moore machine이기 때문에, shift 연산이 수행되기 전 out의 값이 1101이면 z에 1을 넣습니다. 이렇게 순서를 설정하지 않는다면 shift연산이 이루어지고 나서 sequence를 확인하기 때문에 detect가 이뤄지자마자 output이 바로 1으로 바뀝니다.

다음장에 이어서

2.

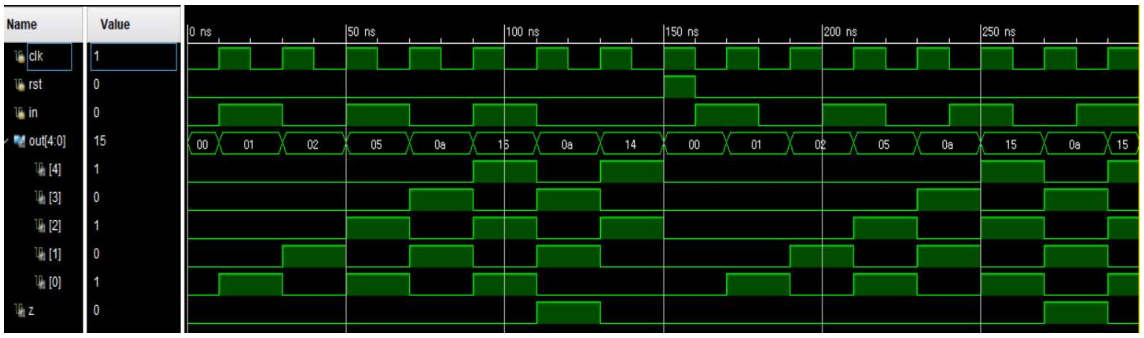
<Sequence Detector 10101 Moore Machine State Diagram>



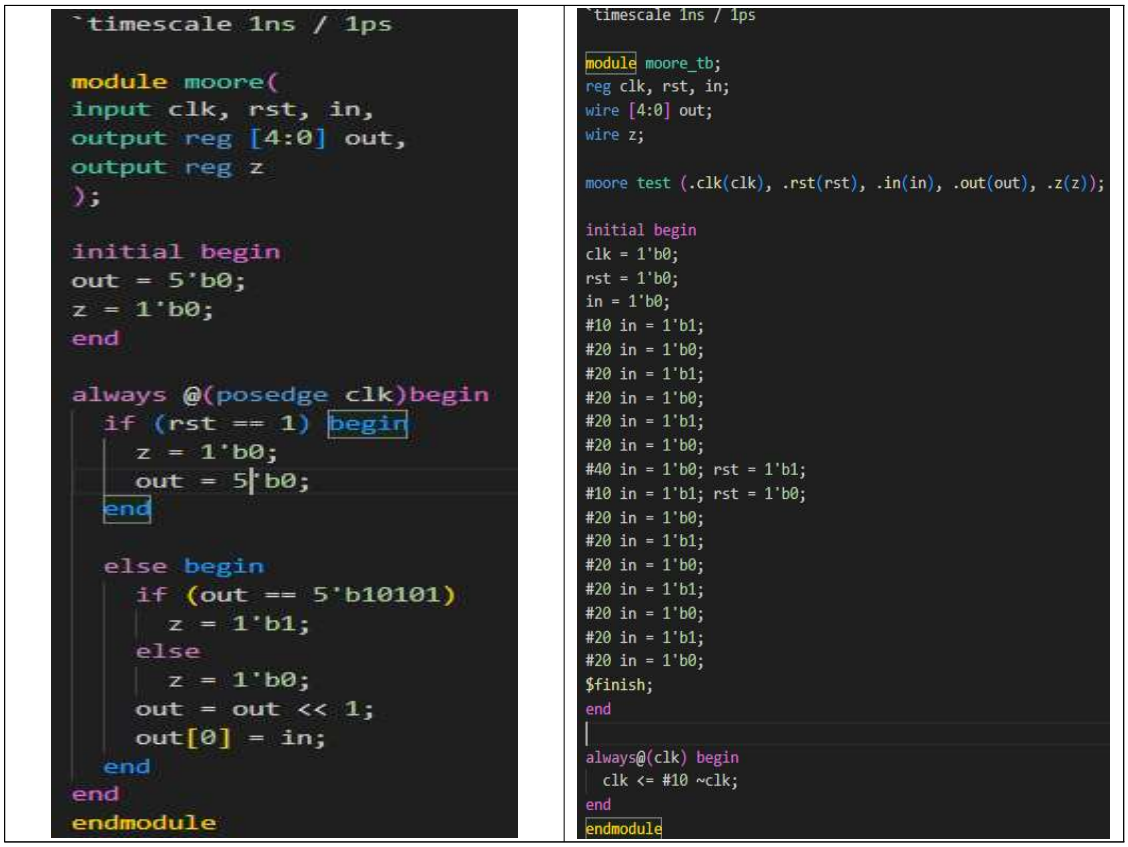
<Sequence Detector 10101 Moore Machine State Table>

q (current state)	q* (next state)		z (output)
	x = 0	x = 1	
A	A	B	0
B	C	B	0
C	A	D	0
D	E	B	0
E	A	F	0
F	E	B	1

Sequence Detector 10101의 Moore machine 상태표에서는 A부터 F까지의 6가지 상태가 존재합니다. F 상태의 경우에만 output인 z가 1으로 출력됩니다. 나머지의 경우는 다 output이 0입니다. A의 상태에서 input이 1이면 B로 전이합니다. B의 상태에서 input이 0이면 C로 전이하고, C의 상태에서는 1이 input으로 들어가면 D로 전이합니다. D의 상태에서는 input이 0이면 E 상태로 전이하고 마지막으로 E의 상태에서 input이 1이면 F 상태로 전이하여 10101의 시퀀스를 감지하여 output으로 1이 출력됩니다.



시뮬레이션 결과에서 알 수 있듯이 10101의 sequence가 차례로 들어왔을 때 바로 output으로 1이 출력되는 것이 아니라 다음 state, 즉 그 다음 clk의 postitive edge에서 출력이 되는 것을 확인할 수 있습니다. 그리고 rst이 1이면 모두 초기화됩니다. out[0]이 가장 최근에 들어온 비트이며 << 연산자를 사용하여 0, 1, 2, 3, 4 순으로 shift되는 모습을 확인할 수 있습니다. 예를 들어 00001이었고 다음 input으로 0이 들어오면 00010이 됩니다.



```
`timescale 1ns / 1ps

module moore(
input clk, rst, in,
output reg [4:0] out,
output reg z
);

initial begin
out = 5'b0;
z = 1'b0;
end

always @(posedge clk)begin
  if (rst == 1) begin
    z = 1'b0;
    out = 5'b0;
  end

  else begin
    if (out == 5'b10101)
      z = 1'b1;
    else
      z = 1'b0;
    out = out << 1;
    out[0] = in;
  end
end
endmodule
```

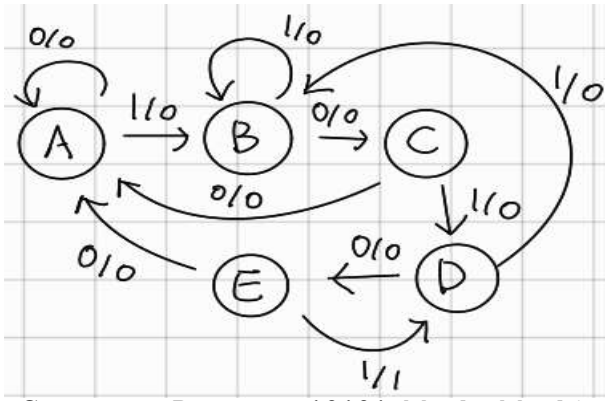
```
`timescale 1ns / 1ps

module moore_tb;
reg clk, rst, in;
wire [4:0] out;
wire z;

moore test (.clk(clk), .rst(rst), .in(in), .out(out), .z(z));

initial begin
clk = 1'b0;
rst = 1'b0;
in = 1'b0;
#10 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#40 in = 1'b0; rst = 1'b1;
#10 in = 1'b1; rst = 1'b0;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
$finish;
end

always@(clk) begin
  clk <= #10 ~clk;
end
endmodule
```

input으로는 clk, rst, in을 선언하고 output으로는 reg형으로 크기 5짜리 배열 [4:0] out과 시퀀스가 탐지되었을 때 1을 출력할 z를 선언해줍니다. 또한 시뮬레이션상 편의를 위해 초기 output 값을 모두 0으로 설정해줍니다. 5개의 out중 out[0]이 가장 최근에 들어온 bit임을 가정하고 코드를 작성합니다.

해당 모듈은 clk의 positive edge에서만 작동하고 rst으로 1이 들어올 경우 5개의 out과 z를 모두 0으로 초기화해줍니다. 그 외의 경우에는 좌측으로 1비트씩 shift 연산(<<)을 하고 가장 최근에 들어온 bit를 저장하는 out[0]에 in값을 넣어줍니다. 또한, moore machine이기 때문에, shift 연산이 수행되기 전 out의 값이 10101이면 z에 1을 넣습니다. 이렇게 순서를 설정하지 않는다면 shift 연산이 이루어지고 나서 sequence를 확인하기 때문에 detect가 이뤄지자마자 output이 바로 1으로 바뀝니다.

다음 장에 이어서

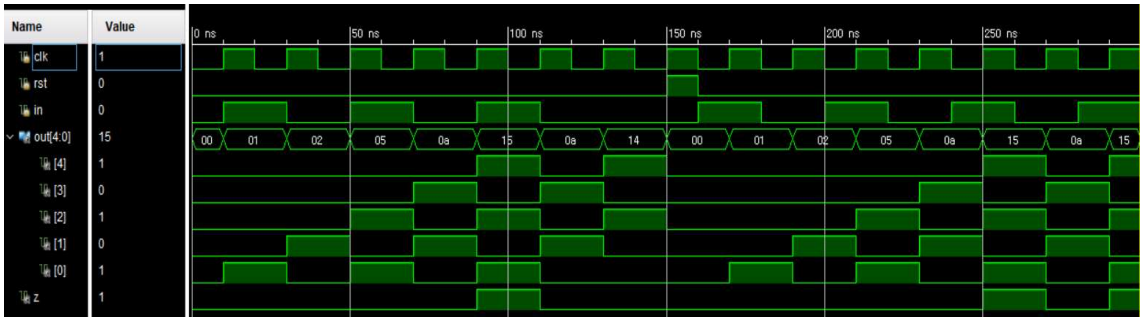
<Sequence Detector 10101 Mealy Machine State Diagram>



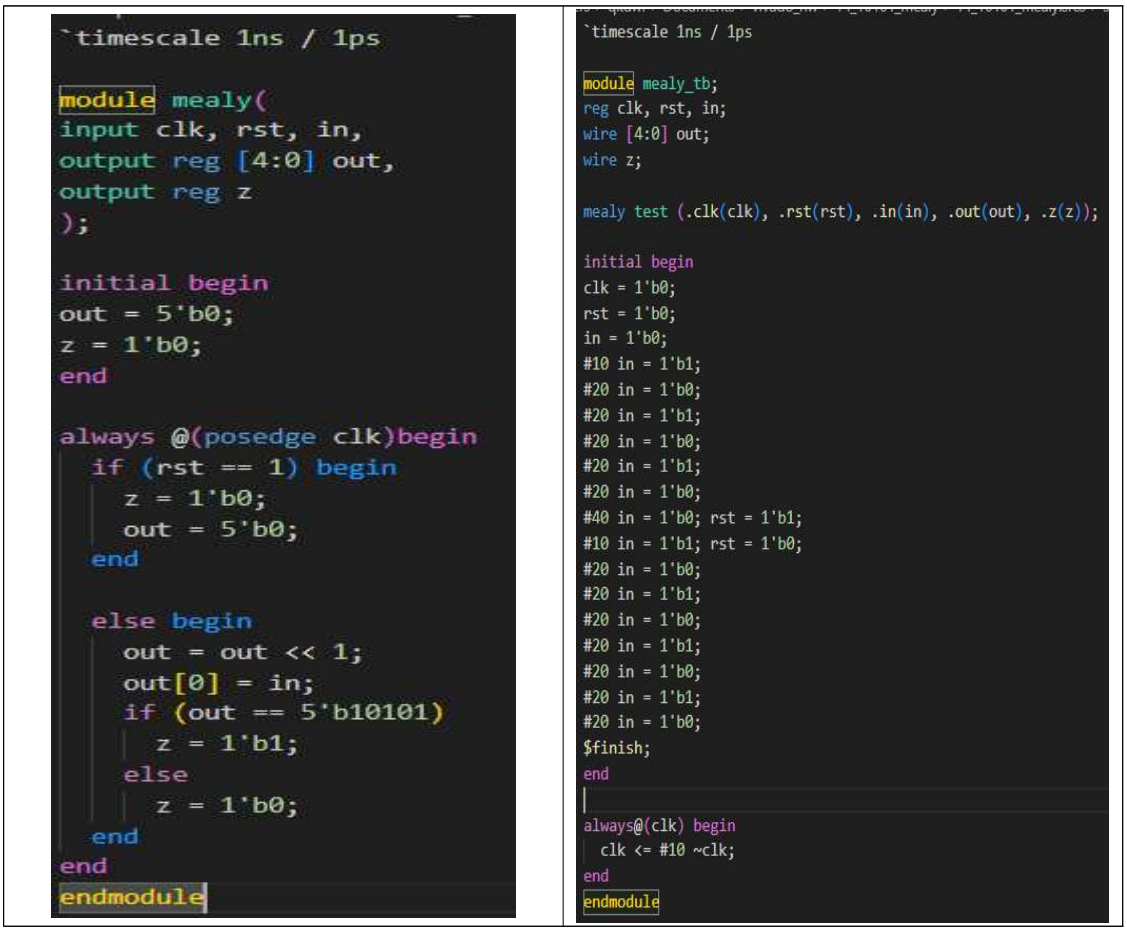
<Sequence Detector 10101 Mealy Machine State Table>

q (current state)	q* (next state)		z (output)	
	x = 0	x = 1	x = 0	x = 1
A	A	B	0	0
B	C	B	0	0
C	A	D	0	0
D	E	B	0	0
E	A	D	0	1

Sequence Detector 10101의 Mealy machine 상태표에서는 Moore 버전의 6개와 다르게 A부터 E까지의 5가지 상태가 존재합니다. E 상태의 경우에만 x의 값으로 1이 입력되면 output인 z가 1으로 출력됩니다. 나머지의 경우는 다 output이 0입니다. A의 상태에서 input이 1이면 B로 전이합니다. B의 상태에서 input이 0이면 C로 전이하고, C의 상태에서는 1이 input으로 들어가면 D로 전이합니다. D의 상태에서는 input이 0이면 E 상태로 전이하고 마지막으로 E의 상태에서 input이 1이면 D 상태로 전이하여 10101의 시퀀스를 감지하여 output으로 1이 출력됩니다.



앞서 설명했던 moore와 다르게 시뮬레이션 결과에서 알 수 있듯이 10101의 sequence가 차례로 들어왔을 때 마지막에서 input으로 1이 들어오기에 바로 output으로 1이 출력되는 것을 확인할 수 있습니다. Mealy machine은 state 뿐만 아니라 현재의 input도 고려하기 때문입니다. 그리고 rst이 1이면 모두 초기화됩니다. 또한, out[0]이 가장 최근에 들어온 비트이며 << 연산자를 사용하여 0, 1, 2, 3, 4 순으로 shift되는 모습을 확인할 수 있습니다. 예를 들어 00001이었고 다음 input으로 0이 들어오면 00010이 됩니다. - 이어서 다음 장에서



```
`timescale 1ns / 1ps

module mealy(
input clk, rst, in,
output reg [4:0] out,
output reg z
);

initial begin
out = 5'b0;
z = 1'b0;
end

always @(posedge clk)begin
  if (rst == 1) begin
    z = 1'b0;
    out = 5'b0;
  end

  else begin
    out = out << 1;
    out[0] = in;
    if (out == 5'b10101)
      z = 1'b1;
    else
      z = 1'b0;
  end
end
endmodule
```

```
`timescale 1ns / 1ps

module mealy_tb;
reg clk, rst, in;
wire [4:0] out;
wire z;

mealy test (.clk(clk), .rst(rst), .in(in), .out(out), .z(z));

initial begin
clk = 1'b0;
rst = 1'b0;
in = 1'b0;
#10 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#40 in = 1'b0; rst = 1'b1;
#10 in = 1'b1; rst = 1'b0;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
#20 in = 1'b1;
#20 in = 1'b0;
$finish;
end

always@(clk) begin
  clk <= #10 ~clk;
end
endmodule
```

input으로는 clk, rst, in을 선언하고 output으로는 reg형으로 크기 5짜리 배열 [4:0] out과 시퀀스가 탐지되었을 때 1을 출력할 z를 선언해줍니다. 또한 시뮬레이션 상 편의를 위해 초기 output 값을 모두 0으로 설정해줍니다. 5개의 out중 out[0]이 가장 최근에 들어온 bit임을 가정하고 코드를 작성합니다.

해당 모듈은 clk의 positive edge에서만 작동하고 rst으로 1이 들어올 경우 5개의 out과 z를 모두 0으로 초기화해줍니다. 그 외의 경우에는 좌측으로 1비트씩 shift 연산(<<)을 하고 가장 최근에 들어온 bit를 저장하는 out[0]에 in값을 넣어줍니다. 또한, mealy machine이기 때문에, shift 연산을 수행하고 나서 out의 값이 10101이면 z에 1을 넣습니다. 이렇게 순서를 설정하여 shift 연산이 이루어지고 나서 sequence를 확인하기 때문에 detect가 이뤄지자마자 해당 clk의 positive edge에서 output이 바로 1으로 바뀝니다.