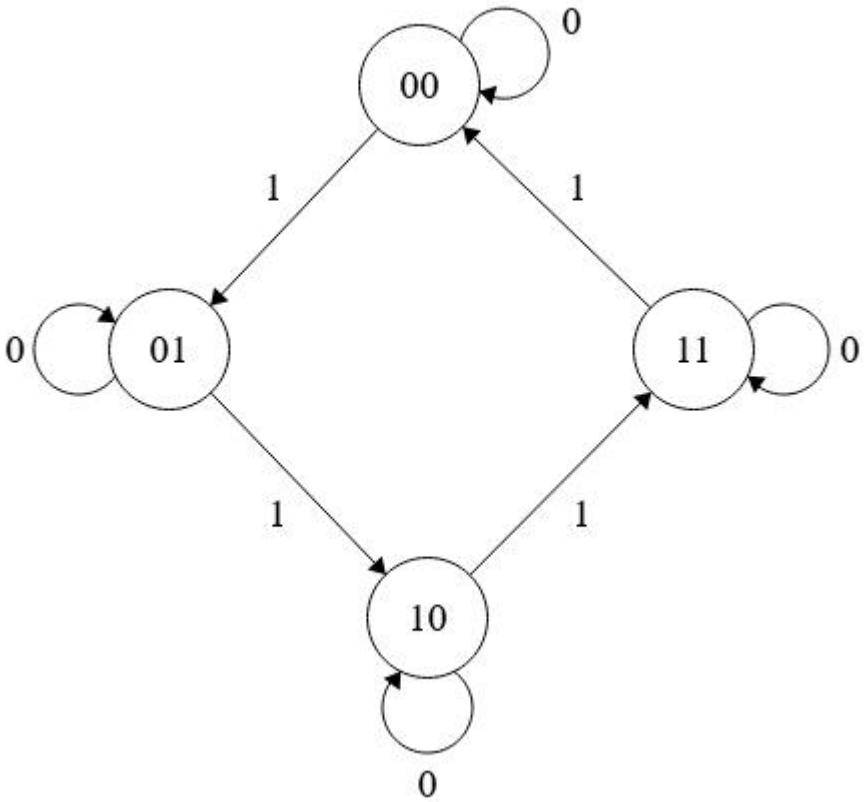
12주차 결과보고서

전공: 신문방송학과 학년: 4학년 학번: 20191150 이름: 전현길

1. 2-bit counter의 결과 및 Simulation 과정에 대해서 설명하시오.

Moore model

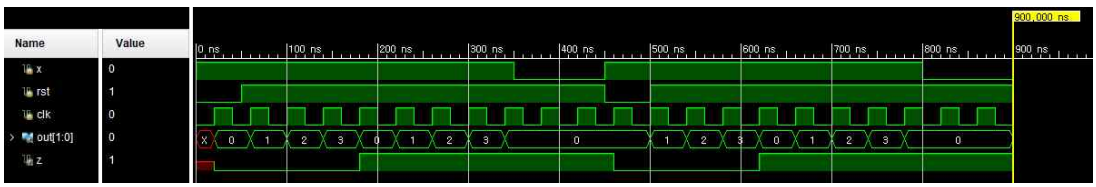


Q(t)	next state Q(t+1)	
	x = 0	x = 1
00	00	01
01	01	10
10	10	11
11	11	00

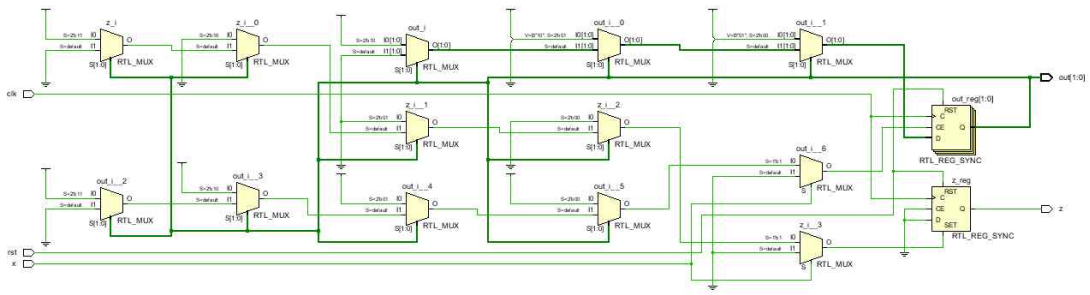
source code	testbench code

`timescale 1ns / 1ps
 	
module bit_counter (
input x, rst, clk,
output reg [1:0] out,
output reg z
);
 	
always @(posedge clk) begin
if (!rst) begin
out <= 2'b00;
z <= 1'b0;
end
else if (x) begin
if(out == 2'b00) out <= 2'b01;
else if(out == 2'b01) out <= 2'b10;
else if(out == 2'b10) out <= 2'b11;
else if(out == 2'b11) begin
out <= 2'b00;
z <= 1'b1;
end
end
end
 	
endmodule
	`timescale 1ns / 1ps
	
	module bit_counter_tb;
	
	reg x, rst, clk;
	wire [1:0] out;
	wire z;
	
	bit_counter u_test(
	.x (x), .rst (rst), .clk (clk),
	.out (out), .z (z)
	);
	
	initial begin
	x = 1'b1; rst = 1'b0; clk = 1'b0; // 시작할 때 값을 reset
	forever #20 clk = ~clk;
	end
	
	initial begin
	#50 rst = 1'b1;
	#300 x = 1'b0;
	#100 rst = 1'b0; x = 1'b1; // x를 0으로 두었을 때 값이 변하는지 확인
	#50 rst = 1'b1;
	#300 x = 1'b0;
	#100
	$finish;
	end
	
	endmodule

2-bit counter simultation



2-bit counter circuit diagram

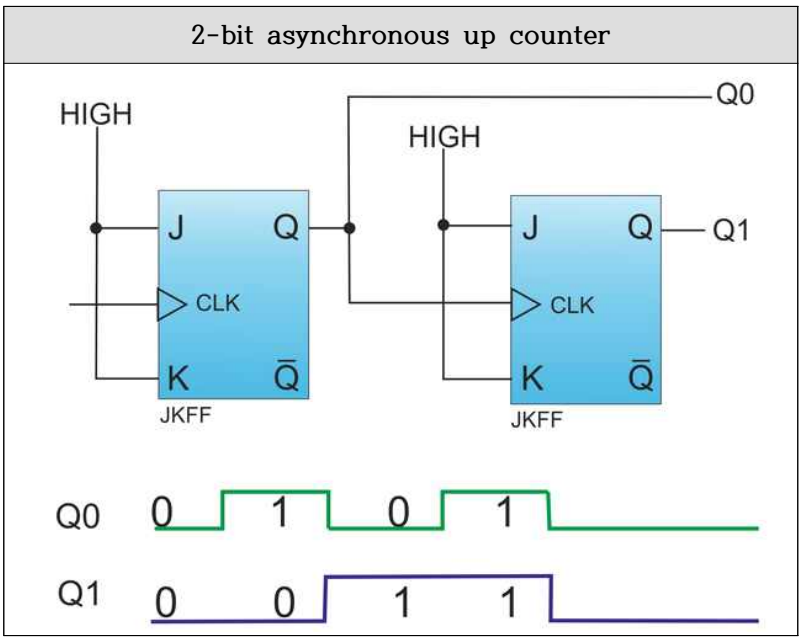


카운터란 데이터 입력이 없이 연속되는 클럭에 따라 정해진 일련의 상태를 반복하는 소자이다. 입력 신호의 주기(pulse)를 세서, 클럭 신호에 따라 값을 증가시키거나 감소하면서 정해진 출력값의 순서를 반복한다.

이번에 구현할 2-bit counter의 경우 **2bit 데이터로 표현할 수 있는 4가지 상태(00, 01, 10, 11)를 순회하는 카운터**이다. 상태표에서 볼 수 있듯이, **입력 값 x = 0일 때 현재 상태를 유지하고 x = 1일 때 다음 상태로 전환**한다. 출력이 오로지 현재의 상태에 의해서만 결정되므로 moore model diagram으로 표현할 수 있다.

Verilog 코드 상의 구현을 위해 always @(posedge clk)문, else if (x) 문을 사용하여 클럭이 상승 엣지일 때 입력값 x의 현재 값에 따라 상태를 변화시키도록 했다. 또, rst 입력이 set되었을 때 모든 카운터의 현재 상태를 초기화시키도록 했다.

Verilog 코드를 상당히 추상화된 수준에서 작성했기 때문에, 실질적인 회로의 형태를 고려하지 않고 상태 변화를 기준으로 코드가 작성되었다. 회로를 조금 더 최적화시키기 위해 세부적인 구현까지 코드로 작성할 경우, JK flip-flop을 먼저 구현한 뒤 이를 2개 연결하는 방식으로 구현할 수 있다. asynchronous 2-bit up counter의 구체적인 구현도는 아래와 같다.



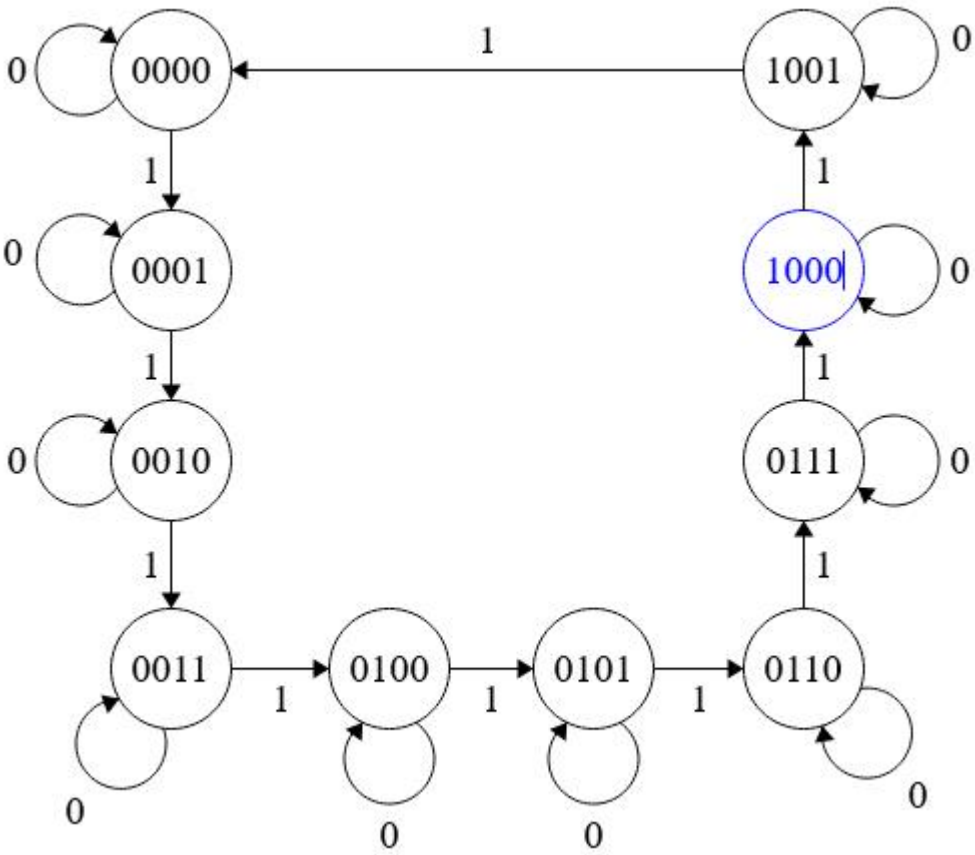
시뮬레이션 결과, rst 값에 따라 회로 상태의 초기화가 정상적으로 이루어졌고, x 값에 따라 x = 0일 때 상태가 고정되었고, x = 1일 때에는 상태가 변화하였다. 또, x = 0, rst = 0일 때에는 정상적인 순서대로 상태가 변화하는 것을 확인할 수 있었다. 회로가 (rst 입력을 생략한) moore diagram과 동일하게 동작하므로 성공적으로 구현된 것을 확인할 수 있었다.

시뮬레이션의 시작 부분의 빨간색 신호는 값이 지정되지 않았음을 나타내는

데, 이는 첫 번째 클럭 신호의 transition이 아직 이뤄지지 않아 값이 정해지지 않았기 때문이므로 구현상의 문제는 없다.

2. 4-bit decade counter의 결과 및 Simulation 과정에 대해서 설명하시오.

Moore model

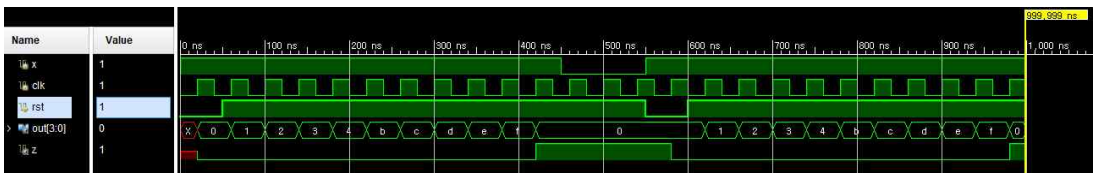


Q(t)	next state Q(t+1)	
	x = 0	x = 1
0000	0000	0001
0001	0001	0010
0010	0010	0011
0011	0011	0100
0100	0100	0101
0101	0101	0110
0110	0110	0111
0111	0111	1000
1000	1000	1001
1001	1001	0000

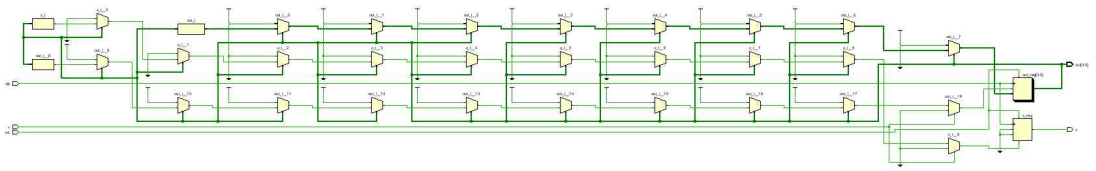
source code	testbench code

`timescale 1ns / 1ps
 	
module dec_counter (
  input x, rst, clk,
  output reg [3:0] out,
  output reg z
);
 	
always @(posedge clk) begin
  if (!rst) begin
  out <= 4'b0000;
  z <= 1'b0;
  end
  else if (x) begin
  if (out == 4'b1001) z = 1'b1;
  out = out + 1;
  if (out == 4'b1010) out = 4'b0000;
  end
end
 	
endmodule
	`timescale 1ns / 1ps
	
	module dec_counter_tb;
	
	reg x, clk, rst;
	wire [3:0] out;
	wire z;
	
	dec_counter u_test(
	  .x (x), .rst (rst), .clk (clk),
	  .out (out), .z (z)
	);
	
	initial begin
	  x = 1'b1; rst = 1'b0; clk = 1'b0; // 시작할 때 값을 reset
	  forever #20 clk = ~clk;
	end
	
	initial begin
	  #50 rst = 1'b1;
	  #400 x = 1'b0;
	  #100 rst = 1'b0; x = 1'b1; // x를 0으로 두었을 때 값이 변하는지 확인
	  #50 rst = 1'b1;
	  #400 x = 1'b0;
	  #100
	  $finish;
	end
	
	endmodule

4-bit 8421 decade counter simultation



4-bit decade counter circuit diagram



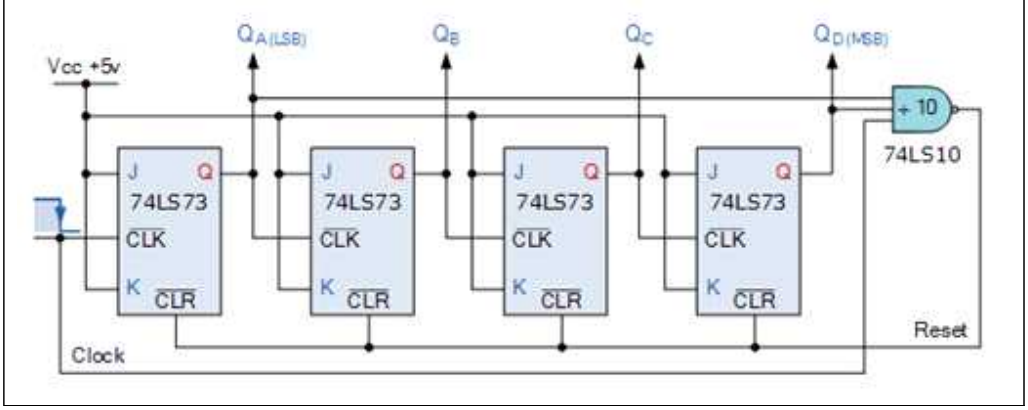
이번에 구현할 4-bit decade counter의 경우 **4bit 데이터로 0-9를 나타내는 8421 BCD 코드(0000-1001)를 순회하는 카운터**이다. 상태표에서 볼 수 있듯이, **입력값 x = 0일 때 현재 상태를 유지하고 x = 1일 때 다음 상태로 전환**한다. 출력이 오로지 현재의 상태에 의해서만 결정되므로 moore model diagram으로 표현할 수 있다.

Verilog 코드 상의 구현을 위해 always @(posedge clk)문, else if (x) 문을 사용하여 클럭이 상승 엣지일 때 입력값 x의 현재 값에 따라 상태를 변화시키

도록 했다. 또, rst 입력이 set되었을 때 모든 카운터의 현재 상태를 초기화시키도록 했다.

Verilog 코드를 상당히 추상화된 수준에서 작성했기 때문에, 실질적인 회로의 형태를 고려하지 않고 상태 변화를 기준으로 코드가 작성되었다. 회로를 조금 더 최적화시키기 위해 세부적인 구현까지 코드로 작성할 경우, JK flip-flop을 먼저 구현한 뒤 이를 4개 연결하는 방식으로 구현할 수 있다. asynchronous decade up counter의 구체적인 구현도는 아래와 같다.

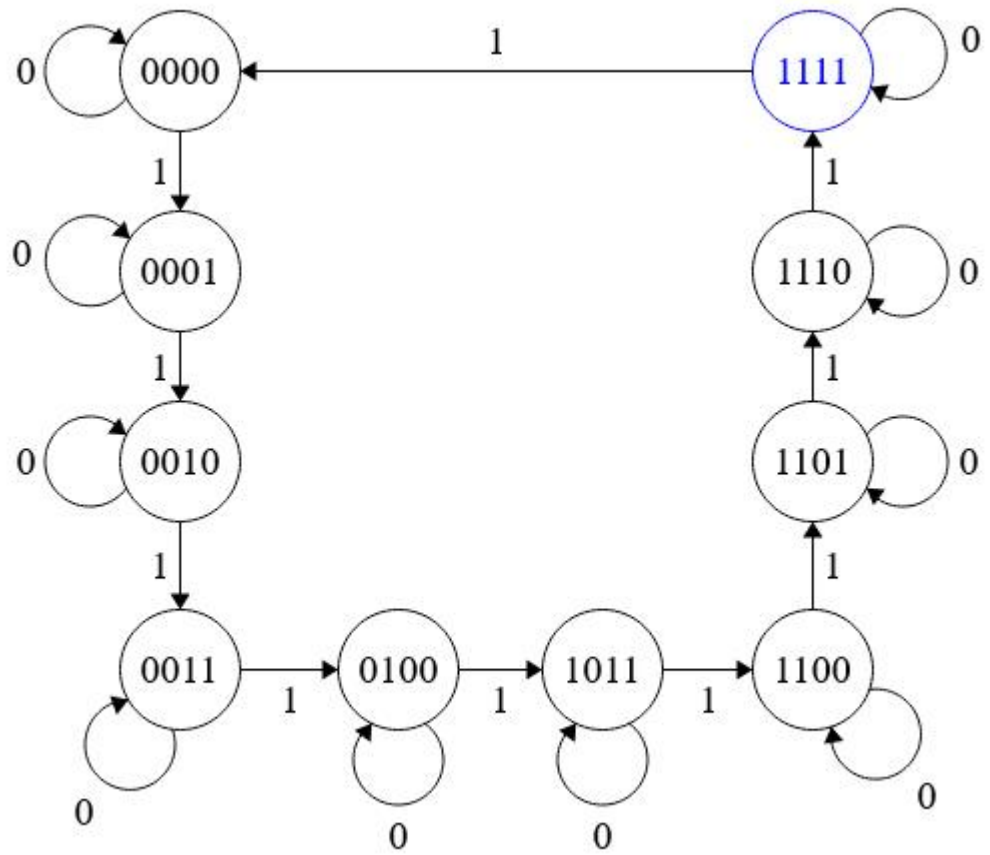
asynchronous decade up counter



시뮬레이션 결과, rst 값에 따라 회로 상태의 초기화가 정상적으로 이루어졌고, x 값에 따라 x = 0일 때 상태가 고정되었고, x = 1일 때에는 상태가 변화하였다. 또, x = 0, rst = 0일 때에는 정상적인 순서대로 상태가 변화하는 것을 확인할 수 있었다. 회로가 (rst 입력을 생략한) moore diagram과 동일하게 동작하므로 성공적으로 구현된 것을 확인할 수 있었다.

3. 4-bit 2421 decade counter의 결과 및 Simulation 과정에 대해서 설명하시오.

Moore model

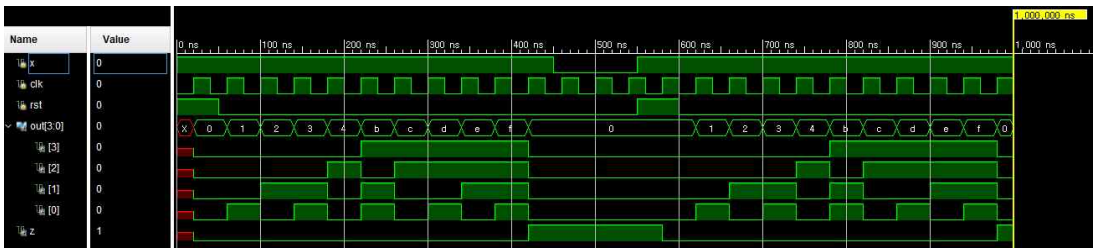


Q(t)	next state Q(t+1)	
	x = 0	x = 1
0000	0000	0001
0001	0001	0010
0010	0010	0011
0011	0011	0100
0100	0100	1011
1011	1011	1100
1100	1100	1101
1101	1101	1110
1110	1110	1111
1111	1111	0000

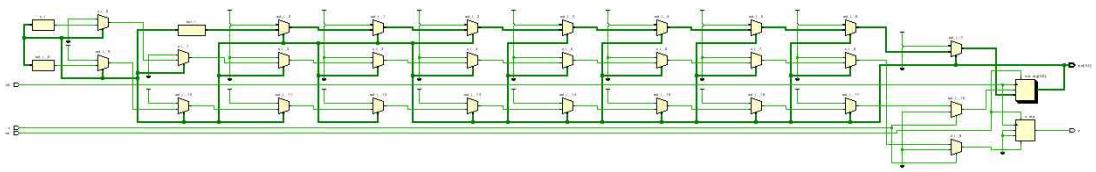
source code	testbench code

`timescale 1ns / 1ps
 	
module dec_counter_2421 (
  input x, rst, clk,
  output reg [3:0] out,
  output reg z
);
 	
always @(posedge clk) begin
  if (!rst) begin
  out <= 4'b0000;
  z <= 1'b0;
  end
  else if (x) begin
  if (out == 4'b1111) z = 1'b1;
  out = out + 1;
  if (out == 4'b0101) out = 4'b
1011;
  end
end
 	
endmodule
	`timescale 1ns / 1ps
	
	module dec_counter_2421_tb;
	
	reg x, clk, rst;
	wire [3:0] out;
	wire z;
	
	dec_counter_2421 u_test(
	  .x (x), .rst (rst), .clk (clk),
	  .out (out), .z (z)
	);
	
	initial begin
	  x = 1'b1; rst = 1'b0; clk = 1'b0; // 시작할 때 값을 reset
	  forever #20 clk = ~clk;
	end
	
	initial begin
	  #50 rst = 1'b1;
	  #400 x = 1'b0;
	  #100 rst = 1'b0; x = 1'b1; // x를 0으로 두었을 때 값이 변하는지 확인
	  #50 rst = 1'b1;
	  #400 x = 1'b0;
	  #100
	  $finish;
	end
	
	endmodule

4-bit 2421 decade counter simultation



4-bit decade 2421 counter circuit diagram



이번에 구현할 4-bit decade counter의 경우 **4bit 데이터로 0-9를 나타내는 2421 BCD 코드(0000-1111)를 순회하는 카운터**이다. 상태표에서 볼 수 있듯이, **입력값 x = 0일 때 현재 상태를 유지하고 x = 1일 때 다음 상태로 전환**한다. 출력이 오로지 현재의 상태에 의해서만 결정되므로 moore model diagram으로 표현할 수 있다.

Verilog 코드를 상당히 추상화된 수준에서 작성했기 때문에, 실질적인 회로의 형태를 고려하지 않고 상태 변화를 기준으로 코드가 작성되었다. 회로를 조금 더 최적화시키기 위해 세부적인 구현까지 코드로 작성할 경우, JK flip-flop을 먼저 구현한 뒤 이를 4개 연결하는 방식으로 구현할 수 있다. asynchronous decade up counter의 구체적인 구현도는 아래와 같다.

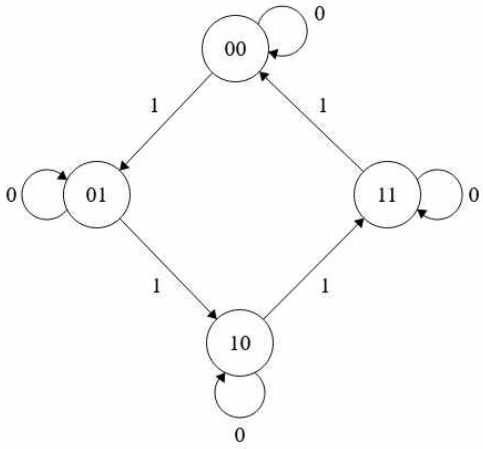
시뮬레이션 결과, rst 값에 따라 회로 상태의 초기화가 정상적으로 이루어졌고, x 값에 따라 x = 0일 때 상태가 고정되었고, x = 1일 때에는 상태가 변화하였다. 또, x = 0, rst = 0일 때에는 정상적인 순서대로 상태가 변화하는 것을 확인할 수 있었다. 회로가 (rst 입력을 생략한) moore diagram과 동일하게 동작하므로 성공적으로 구현된 것을 확인할 수 있었다.

4. 결과 검토 및 논의 사항

이번 실험에서는 구현에 JK flip-flop이 이용되는 2-bit counter, 4-bit decade counter, 4-bit 2421 decade counter를 Verilog 코드로 구현해 보았으며, FPGA 보드를 활용하여 실제 출력 결과를 확인해 보았다. 실험 결과, 순차 회로 방식의 구현에 따라 클럭 신호의 첫 번째 transition 이전까지 값이 할당되지 않는 것을 제외하고는 기대되었던 동작이 정상적으로 이루어졌음을 확인할 수 있었다.

5. 추가 이론 조사 및 작성

조합 회로를 설계하기 위해서는 원하는 입력값에 대해 원하는 출력값을 지정한 진리표를 그리고, 해당 진리표의 각 출력 bit를 karnaugh map, quine-mccluskey method 등을 이용해 간소화시키는 것으로 충분했다. 하지만 순차 회로는 상태(state)가 추가되므로, **입력에 더해 현재 상태까지 고려해야 한다.** 이를 위해서는 **상태표(state table), 상태도(state diagram), 특성표(characteristic table)나 여기표(excitation table)**에 대해 이해해야 한다.

state table	state diagram
	

상태표(state table)는 입력과 현재 상태에 따라, 다음 상태 및 출력이 어떻게 변화하는가를 표시한 표이다. m개의 flip-flop과 n개 입력을 갖는 순차논리회로의 경우 2^{m+n}개의 행을 갖는다. 당연하지만, flip-flop 하나당 1bit를 저장하기 때문에 각 flip-flop은 상태의 bit 수와 대응된다.

상태도(state diagram) 역시 입력과 현재 상태에 따라, 다음 상태 및 출력이 어떻게 변화하는가를 표시한다. 입력과 현재 상태에 따라 출력이 변한다면 moore machine, 현재 상태에 따라서만 출력이 변한다면 mealy machine이고, 각각 표현 방법이 다르다.

JK flip-flop operation								
Characteristic table				Excitation table				
J	K	Comment	Q_{next}	Q	J	K	Comment	Q_{next}
0	0	hold state	Q	0	0	X	No Change	0
0	1	reset	0	0	1	X	Set	1
1	0	set	1	1	X	1	Reset	0
1	1	toggle	$\overline{Q}$	1	X	0	No Change	1

특성표(characteristic table)나 **여기표(excitation table)**는 주로 플립플롭 등의 순차 회로를 설명하기 위한 표이다. 왼쪽에 현재 상태가 bit 단위로 입력된다면 excitation table, 그렇지 않고 현재 상태가 Q로 생략되어 입력된다면

characteristic table이다.

[표 10-15] JK 플립플롭을 이용한 순차 논리회로의 여기표

입력	현재 상태		다음 상태		플립플롭 입력				출력
x	A	B	A	B	J_A	K_A	J_B	K_B	y
0	0	0	1	1	1	X	1	X	0
0	0	1	0	0	0	X	X	1	0
0	1	0	0	1	X	1	1	X	0
0	1	1	1	0	X	0	X	1	0
1	0	0	0	0	0	X	0	X	0
1	0	1	0	1	0	X	X	0	1
1	1	0	1	0	X	0	0	X	0
1	1	1	1	1	X	0	X	0	1

결과적으로 순차 회로를 구현하기 위해서는 다음과 같은 순서를 따른다.

1) **입력, 현재 상태에 따른 출력 및 다음 상태를 결정해 상태표를 그린다.**

2) 각 상태 bit마다 사용할 **플립플롭을 결정**한다.

3) 각 플립플롭에 **입력과 현재 상태 bit가 입력되었을 때, 현재 상태 bit를 원하는 다음 상태로 만들기 위해 필요한 입력**을 표로 작성한다.

4) 각 플립플롭 입력 및 출력을 bit 단위로 나누어, 논리식을 작성하고 간소화한 뒤 회로로 구현한다. 이 작업부터는 조합 회로와 동일하다.