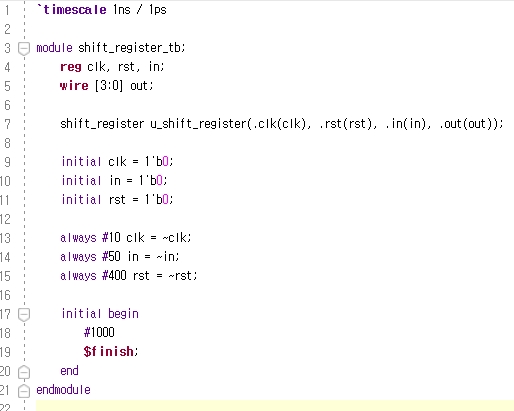
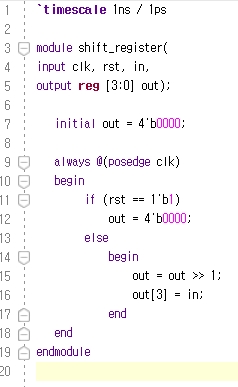
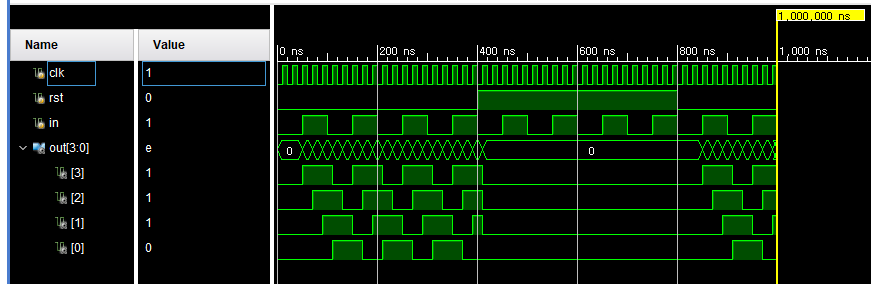
13주차 결과보고서

전공: 기계공학과 학년: 3학년 학번: 20191820 이름: 김형준

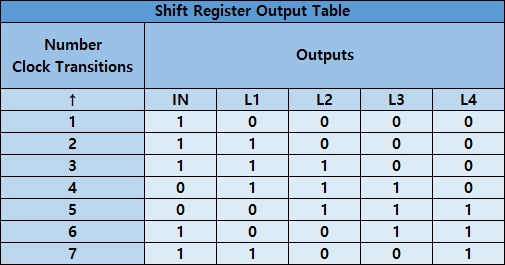
**1.**



[4bit Shift Register의 Verilog Code]



[4bit Shift Register의 Simulation 결과]



[4bit Shift Register의 Output Table]

Shift Register는 입력받은 데이터를 플립플롭에 저장하고, 저장된 데이터를 매 클록마다 한 칸씩 이동시킨다. 이번 실험에서 설계한 4bit Shift Register는 초기상태가 이고, 클록의 rising edge마다 입력받은 데이터(IN)를 한 칸씩 오른쪽(>>)으로 이동시킨다.

위 코드에서는 clk가 posedge(rising edge)일 때 trigger 되고, 4비트의 out을 먼저 으로 초기화 시킨다.

rst(reset)신호가 있을 때(rst=1) trigger 되면 레지스터에 저장된 모든 데이터를 으로 초기화 하고, rst신호가 없을 때 trigger 되면 레지스터에 저장된 모든 데이터를 한 칸씩 오른쪽(>>)으로 이동시키고 레지스터의 첫 번째 플립플롭(out[3])에 입력값(in)을 저장시킨다.

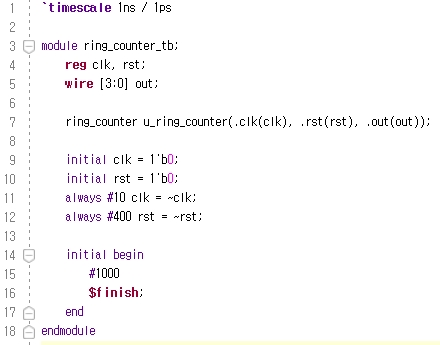
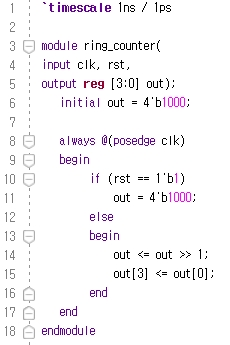
이때, right shift 연산을 사용해 out의 데이터를 이동시키는 연산을 구현했다(맨 오른쪽의 플립플롭(out[0])의 값은 버려짐).

위 시뮬레이션에서 0~400ns까지는 rst가 0이므로 trigger 될 때마다 out의 값이 한 칸씩 오른쪽으로 움직이지만, 400~800ns 에서는 rst가 1이므로 out의 값이 항상 으로 초기화 되는 것을 확인 할 수 있다.

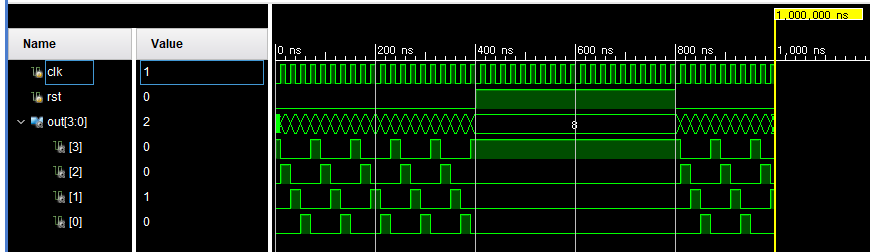
위 Output Table은 위 시뮬레이션에서 첫 번째로 입력값 in의 값이 1이 된 시점(50ns)부터 7번 trigger되었을 때의 입력값과 trigger 직전의 Output을 나타낸 표이다. 시뮬레이션의 0~50ns에서는 in의 값이 0이므로 out이 이미 이라 out의 값의 변화가 없고, 50ns부터 output table에 나타낸 것과 같이 out의 값이 변화하는데, 초기 상태가 인 상황에서 in의 값이 1~3, 6~7번째 trigger에서 1, 4~5번째 trigger에서는 0이므로 각각의 trigger에서 로 변하는 것을 확인할 수 있다. (out에 저장된 값이 한 칸씩 오른쪽으로 이동하고 in의 값이 맨 왼쪽(out[3])에 들어옴)

이후, rst가 1이 되는 400ns까지 out은 100ns를 주기로 의 순서로 clk의 posedge마다 값이 바뀌게 된다.

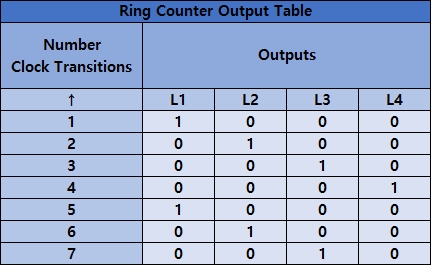
**2.**



[4bit Ring Counter의 Verilog Code]



[4bit Ring Counter의 Simulation 결과]



[4bit Ring Counter의 Output Table]

Ring Counter는 임의의 시간에서 한 개의 플립플롭만 출력이 1이 되고, 저장된 데이터를 매 클록마다 한 칸씩 회전시키는 회로이다. 이번 실험에서 설계한 4bit Ring Counter는 초기상태가 이고, 클록의 rising edge마다 out을 한 칸씩 오른쪽(>>)으로 회전시킨다.

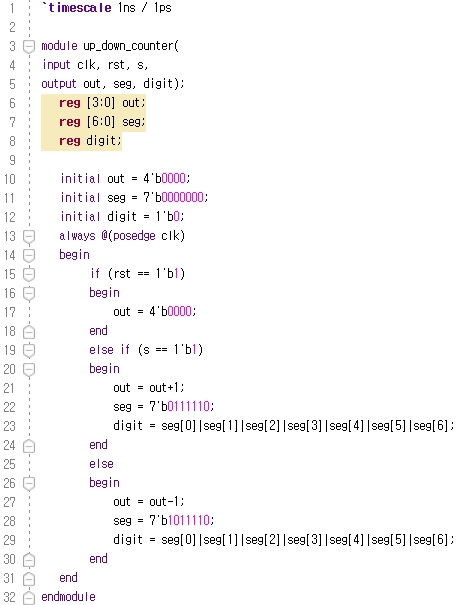
위 코드에서는 clk가 posedge(rising edge)일 때 trigger 되고, 4비트의 out을 먼저 으로 초기화 시킨다.

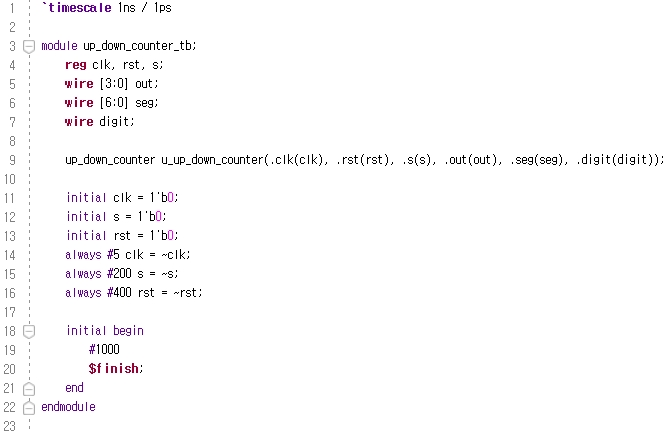
rst(reset)신호가 있을 때(rst=1) trigger 되면 카운터에 저장된 값을 으로 초기화 하고, rst신호가 없을 때 trigger 되면 카운터에 저장된 값을 한 칸씩 오른쪽(>>)으로 이동시키고 레지스터의 첫 번째 플립플롭(out[3])에 마지막 플립플롭(out[0])의 값을 저장시킨다. 이때, right shift 연산을 사용해 out의 데이터를 이동시키는 연산을 구현했으며, non-blocking assignment(<=)를 사용해 out의 right shift 연산과 관계 없이 out[3]에 out[0]의 값(right shift 연산 이전의 값)을 저장할 수 있도록 구현했다. 따라서 위 코드의 out의 값을 이동시키는 부분은 결과적으로 out의 값을 한 칸씩 오른쪽으로 회전시키는 연산을 수행하게 된다.

위 시뮬레이션에서 0~400ns까지는 rst가 0이므로 trigger 될 때마다 out의 값이 한 칸씩 오른쪽으로 회전하지만, 400~800ns 에서는 rst가 1이므로 out의 값이 항상 으로 초기화 되는 것을 확인 할 수 있다.

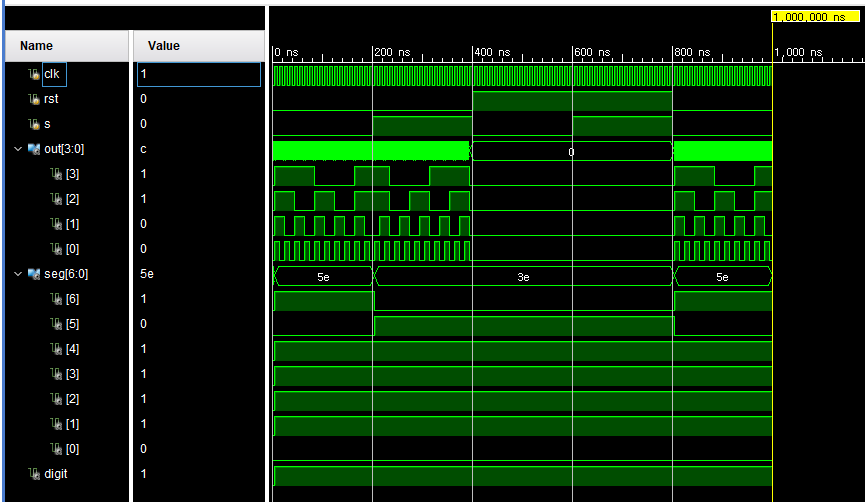
위 Output Table은 위 시뮬레이션에서 60ns부터 7번 trigger되었을 때의 Output을 나타낸 표이다. 위 시뮬레이션에서, out이 로 시작하므로 rst가 1이 되는 400ns까지 80ns를 주기로 trigger가 발생할 때마다 의 순서로 out의 값이 변하는 것을 확인할 수 있다. (out에 저장된 값이 한 칸씩 오른쪽으로 회전함)

**3.**

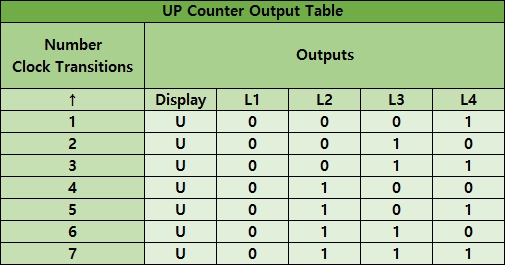


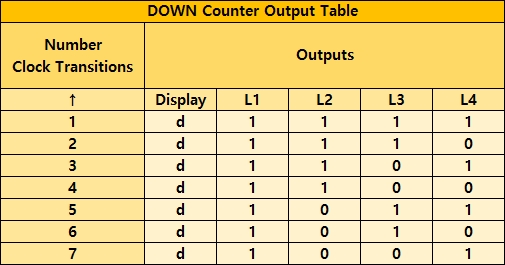


[4bit Up/Down Counter의 Verilog Code]



[4bit Up/Down Counter의 Simulation 결과]





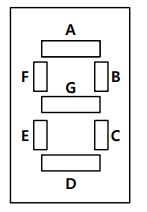
[4bit Up/Down Counter의 Output Table]

Up/Down Counter는 입력값에 따라 counter의 값을 1씩 증가시키는 Up Counter와 1씩감소시키는 Down Counter의 기능을 선택하여 동작할 수 있는 회로이다. 이번 실험에서 설계한 4bit Up/Down Counter는 초기상태가 이고, 클록의 rising edge마다 out의 값을 1씩 증가(s=1), 또는 1씩 감소(s=0) 시킨다.

위 코드에서는 clk가 posedge(rising edge)일 때 trigger 되고, 4비트의 out을 먼저 으로 초기화 시킨다. (7segment LED의 출력을 나타내는 seg(A~G의 출력여부)와 digit(7segment LED 출력 여부, A~G 중 하나 이상 출력시 1)도 각각 0으로 초기화함)

rst(reset)신호가 있을 때(rst=1) trigger 되면 카운터에 저장된 값(out)을 으로 초기화 하고, rst신호가 없을 때 trigger 되면 s의 값에 따라 out의 값을 변경하는데,

s의 값이 1이면 4bit Up Counter로 동작하게 되어 out의 값을 1 증가시키고 (에서 1 증가시 이 되므로 별도로 처리가 필요없음), s의 값이 0이면 4bit Down Counter로 동작하게 되어 out의 값을 1 감소시킨다 (에서 1 감소시 가 되므로 별도로 처리가 필요없음).

이때, 4bit Up Counter로 작동시 7segment로 U를 표시하고 4bit Down Counter로 작동시 7segment로 d를 표시해야 하므로, s가 1일 때 out 증가시 seg에 (B,C,D,E,F 출력)을, digit에 7segment LED 출력여부를 저장한다. s가 0일 때 out 감소시 seg에 (B,C,D,E,G 출력)을, digit에 7segment LED 출력여부를 저장한다. (위 두 경우에서 7segment의 A~G중 출력하는 LED가 있으므로 digit은 모두 1이 됨)

위 시뮬레이션에서 0~400ns까지는 rst가 0이므로 trigger 될 때마다 out의 값이 1씩 변하지만, 400~800ns 에서는 rst가 1이므로 out의 값이 항상 으로 초기화 되는 것을 확인 할 수 있다. 또한, 0~200ns까지는 s가 0이므로 4bit Down Counter로, 200~400ns에서는 s가 1이므로 4bit Up Counter로 작동하는 것을 확인할 수 있다.

위 Output Table은 초기상태가 일 때부터 7번 trigger되었을 때의 Output을 나타낸 표이다. 시뮬레이션의 0ns~200ns에서는 4bit Down Counter로 작동하고, 초기 상태가 이므로 16번의 trigger를 주기로 으로 out의 값이 변하는 것을 확인할 수 있다.

이후, 200ns부터 rst가 1이 되는 400ns까지 4bit Up Counter로 작동하고, 초기상태가 이므로 16번의 trigger를 주기로 으로 out의 값이 변하는 것을 확인할 수 있다.

또한, seg의 값은 s=0인 상태에서 첫번째 trigger 발생부터 s=1로 바뀌고 첫번째 trigger 발생까지(5ns~205ns) (‘d’출력)을, s=1인 상태에서 첫번째 trigger 발생부터 rst=1까지(205ns~400ns) (‘U’출력)을 갖는 것을 알 수 있고, rst = 1일 때는 seg의 값이 그대로 유지되는 것을 확인할 수 있다.

마지막으로, digit은 첫 trigger 이후에는 항상 값이 1로 유지되는 것을 확인할 수 있다.

**4.**

이번 실험을 통해 Shift Register, Ring Counter, Up/Down Counter의 원리와 작동방식에 대해 알 수 있었다. 실험에서 구현한 각각의 레지스터(카운터)에서 clk가 posedge(rising edge)일 때 trigger되게 만들었는데, posedge 대신 negedge를 사용하여 clk가 falling edge일 때에 trigger 되게 만들 수 있다.

Shift Register와 Ring Counter를 구현할 때, out의 값을 이동(회전)시킬 때 Left Shift 연산 또는 Right Shift 연산 중 어느 것으로 구현하느냐에 따라 out에 저장된 데이터의 이동(회전)방향이 왼쪽(Left Shift) 또는 오른쪽(Right Shift)로 결정된다는 것을 알게 되었다.

특히, Up/Down Counter처럼 Shift Register와 Ring Counter에 입력 신호(s)를 하나 더 받을 수 있게 설계해 s의 값에 따라 데이터의 이동 방향을 선택할 수 있게(bidirectional shift register 등) 만들 수도 있다는 것을 알게 되었다.

또한, Ring Counter에서 데이터의 회전을 구현할 때 non-blocking assignment(<=)를 사용해 구현했는데, reg 타입 temp 변수 하나를 추가로 선언하여

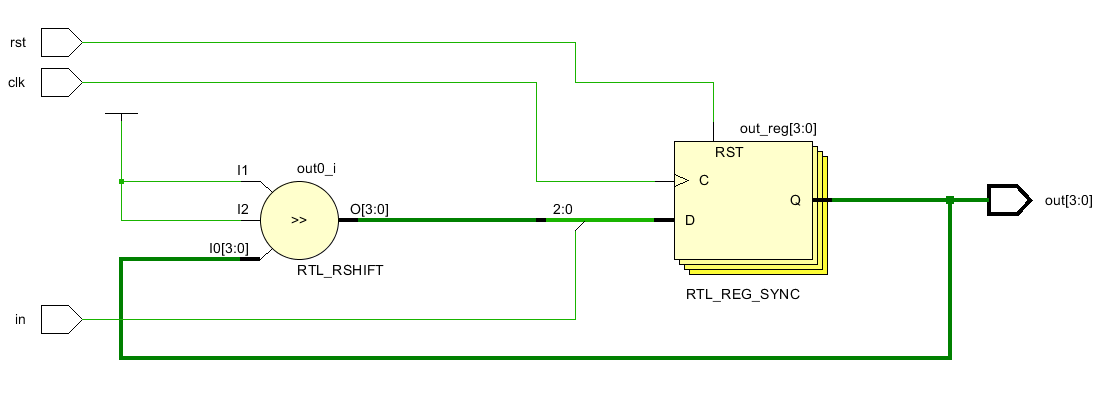
temp = out[0];

out = out >> 1;

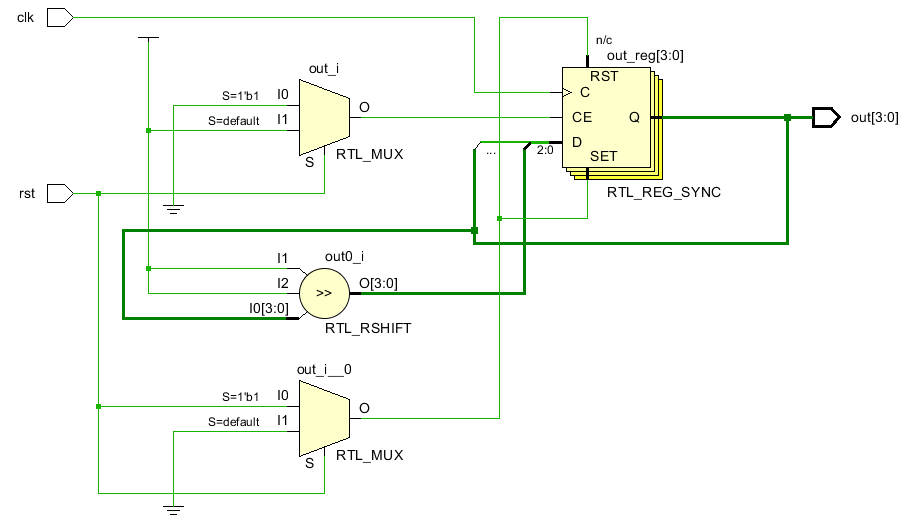
out[3] = temp;

위와 같은 방식으로 blocking assignment으로도 동일한 기능을 구현할 수 있다.

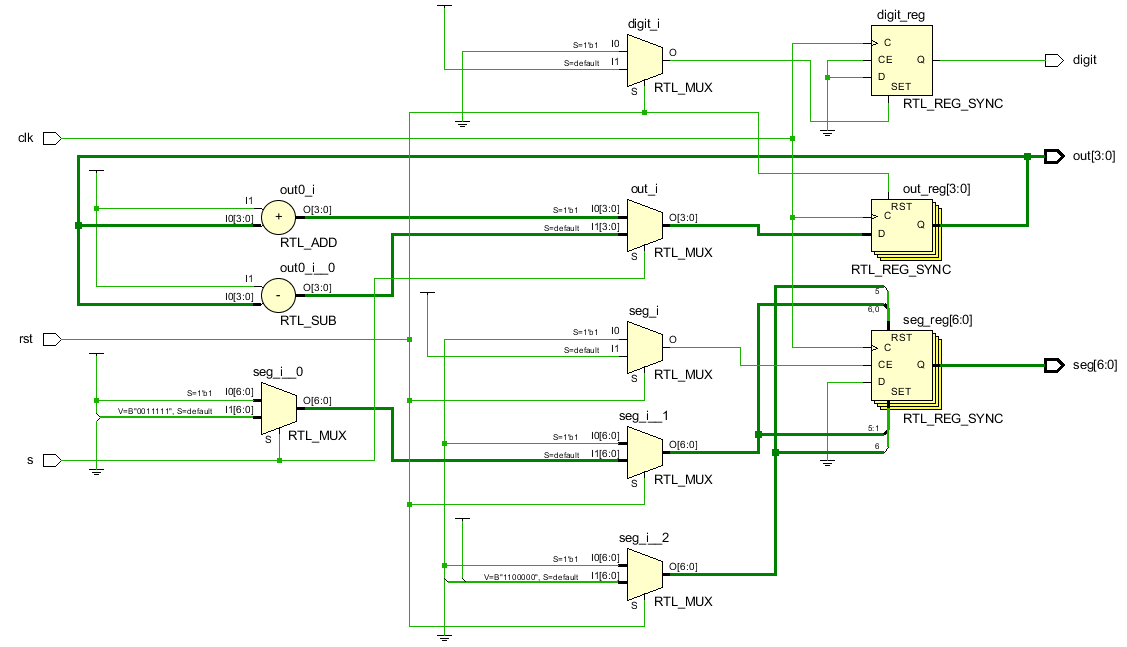
아래의 사진들은 1. ~ 3. 각각의 Schematic이다.



[4bit Shift Register의 Schematic]



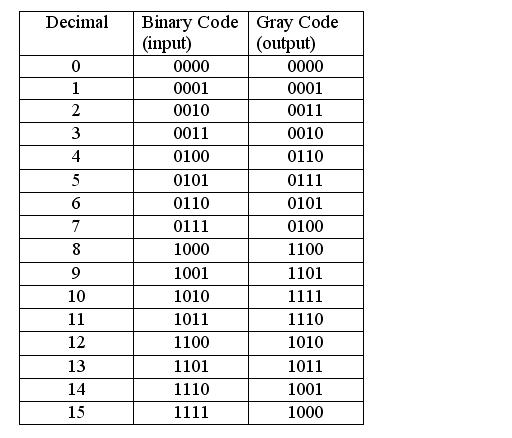
[4bit Ring Counter의 Schematic]



[4bit Up/Down Counter의 Schematic]

**5.**

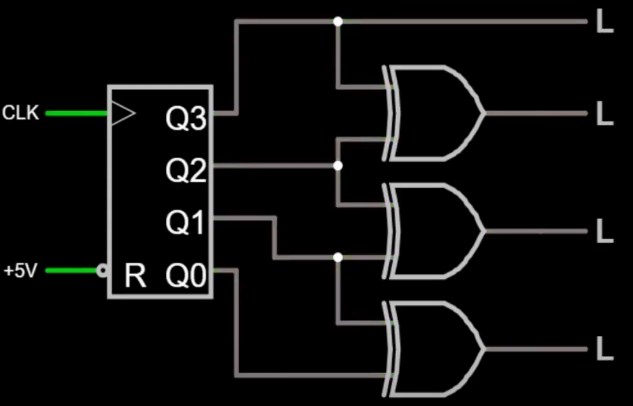
- Gray Code Counter

그레이 코드(Gray Code)는 이진법 표기법 중 하나로, 인접한 수 사이에 한 자리(1bit)만 변하게 만들어진 코드이다. 수의 크기가 변할 때 1bit만 변하면 되므로 하드웨어적인 오류가 적으며, 바꿔야 할 비트의 개수가 적으므로 효율적이다.

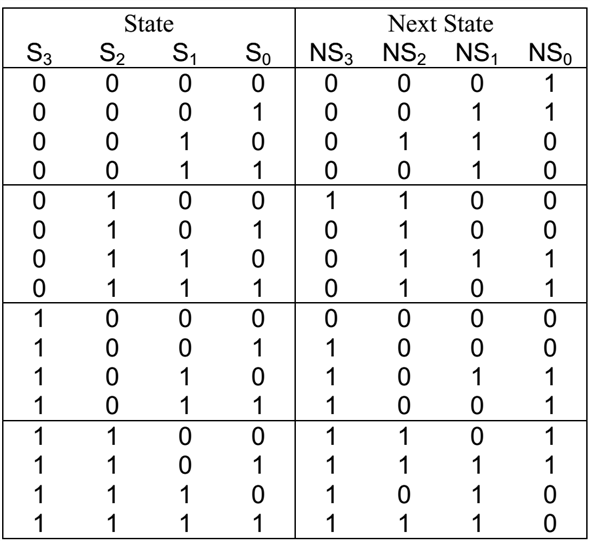
n비트의 그레이 코드의 경우, 개의 서로 다른 수를 표현할 수 있으며, I/O장치, A/D변환기, 주변장치 등에서 숫자를 표현할 때 주로 사용된다.

Gray Code Counter는 이 그레이 코드를 카운터의 상태 값으로 가지며, 매 클럭 펄스마다 Gray Code를 기준으로 카운터의 상태값이 1씩 증가하게 된다(Up Counter의 경우).

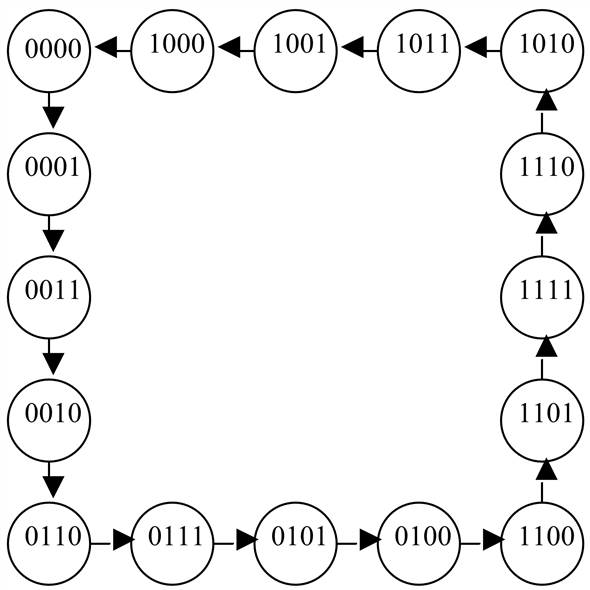
아래는 4bit Gray Code Counter의 회로도, state table, state diagram이다.



[4bit Gray Code Counter의 회로도]



[4bit Gray Code Counter의 state table]



[4bit Gray Code Counter의 state diagram]