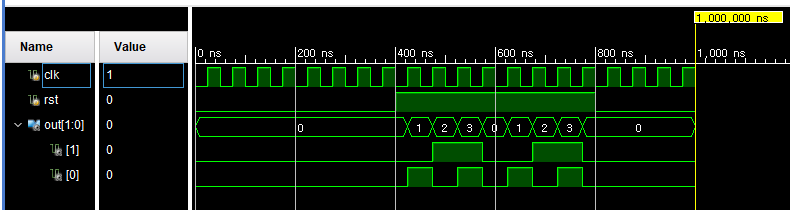
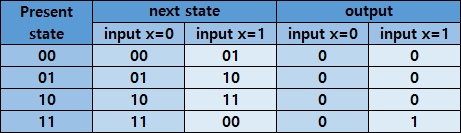
12주차 결과보고서

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

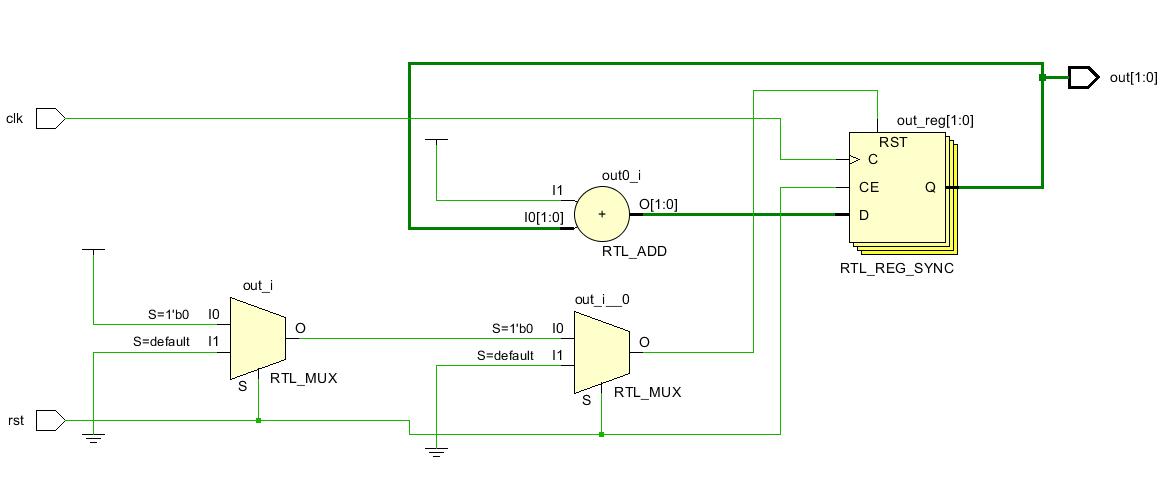
**1.**



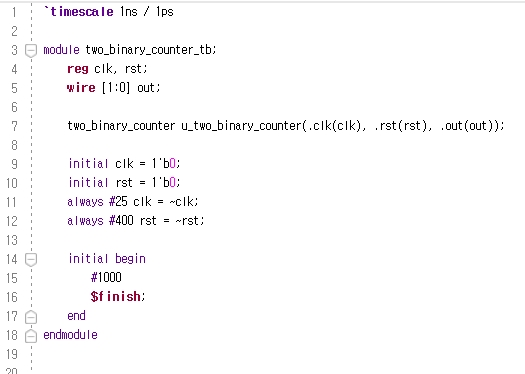
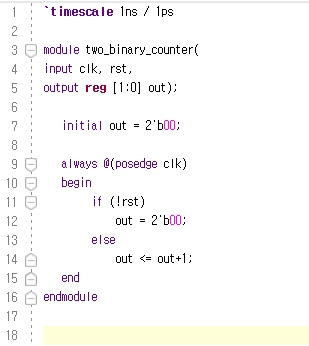
[2bit counter의 Simulation 결과]



[2bit counter의 state table]



[2bit counter의 schematic]

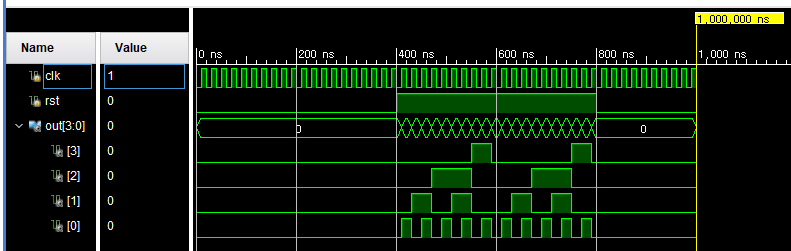


[2bit counter의 Verilog Code]

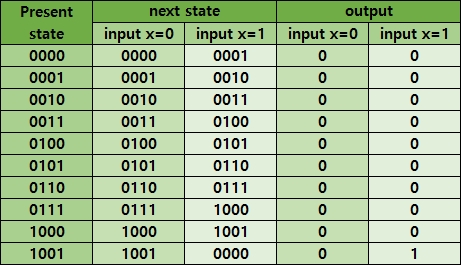
2bit counter는 trigger 될 때마다 , , , 의 상태를 반복하는 회로로, 위 코드에서는 clk가 posedge(rising edge)일 때 counter가 trigger 된다.

위 코드에서는 2비트의 out을 먼저 으로 초기화 시킨다. 그 다음, rst(reset)신호가 있을 때(rst=1) trigger 되면 out의 값에 1을 더해 out이 다음 상태로 넘어가고, rst(reset)신호가 없을 때(rst=0) trigger 되면 out의 값을 으로 초기화 시킨다. 예를 들어 rst 신호가 있을 때 clk가 posedge이면 out이 각각 , , 일 때 , , 이 되고, 일 때는 1을 더하면 이 되지만, out의 크기가 2비트이므로 맨 앞자리의 1이 버려져 이 저장된다. 또한, rst 신호가 없을 때 clk가 posedge이면 out의 값과 상관 없이 out의 값이 으로 초기화 된다. 오른쪽의 test bench 코드는 clk를 25ns마다 보수를 취하고, 400ns마다 rst에 보수를 취함으로써, 2bit counter가 가질 수 있는 모든 경우의 수를 고려할 수 있도록 하였고, (2bit counter의 모든 상태 개수)이므로 rst의 값이 바뀌기 전에 counter의 모든 상태가 한 번이상 순환될 수 있게 했다. 위 시뮬레이션에서 0~400ns 까지는 rst 신호가 없을 때를 나타내어 out의 값이 으로 유지되는 것(trigger될 때마다 으로 초기화)을 볼 수 있고, 400ns~800ns까지는 rst 신호가 있어 trigger될 때 마다 counter의 값이 ~을 순환하는 것을 볼 수 있다.

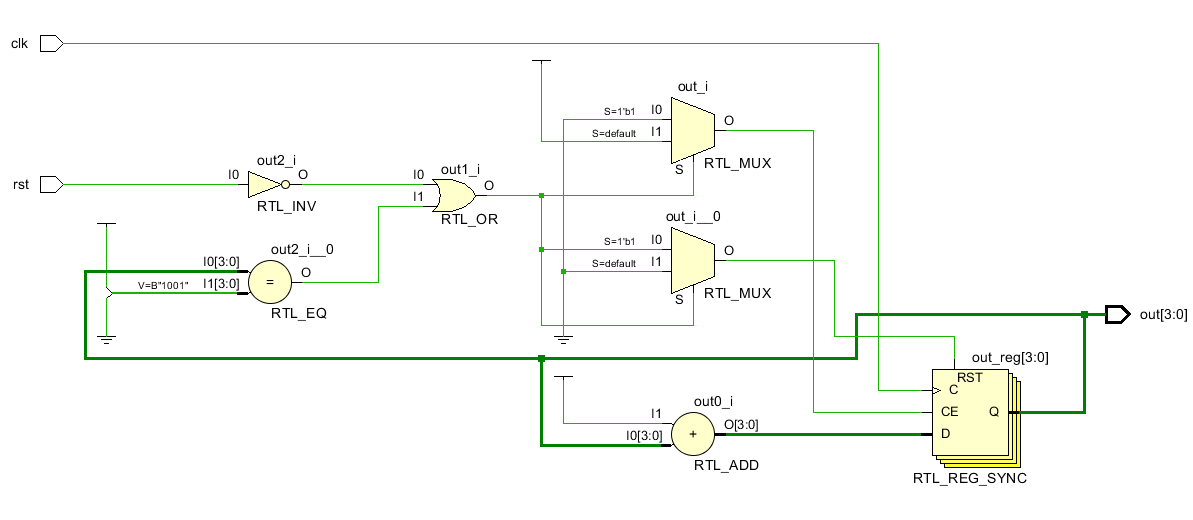
**2.**



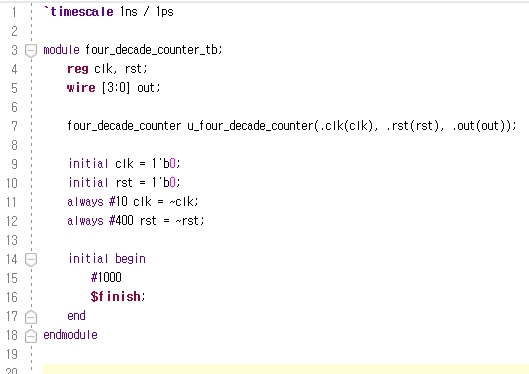
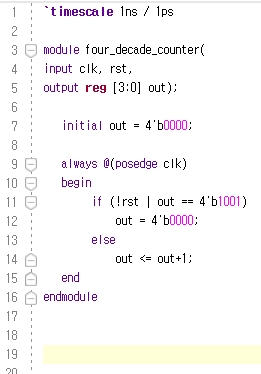
[4bit decade counter의 Simulation 결과]



[4bit decade counter의 state table]



[4bit decade counter의 schematic]

****

[4bit decade counter의 Verilog Code]

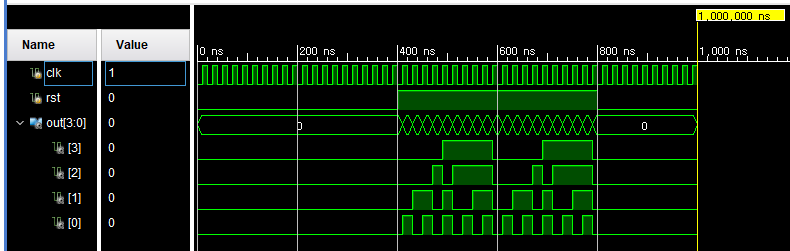
4bit decade counter는 4bit (8421) BCD 코드를 나타내는 counter로, trigger 될 때마다 , , , , , , , , , 의 상태를 반복하는 회로이다. 4bit decade counter는 위의 state table과 같이 10개의 상태를 가지고, 생략된 ~의 상태들은 모두 (8421) BCD 코드에서와 같이 유효하지 않은 값이다.

위 코드에서는 clk가 posedge(rising edge)일 때 counter가 trigger 된다.

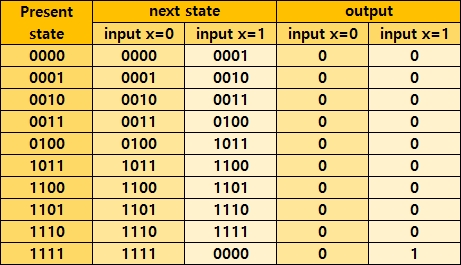
위 코드에서는 4비트의 out을 먼저 으로 초기화 시킨다. 그 다음, rst(reset)신호가 신호가 없거나(rst=0), out의 값이 (9)일 때 trigger 되면 out의 값을 으로 초기화 시킨다. rst 신호가 있을 때(rst=1) trigger 되면 out의 값에 1을 더해 out이 다음 상태로 넘어간다.

예를 들어 rst 신호가 있을 때 clk가 posedge이면 out이 각각 , , , , , , , , 일 때 , , , , , , , , 이 되고, 일 때는 으로 초기화 된다. (에 1을 더하면 이 되어 2bit counter처럼 자동으로 초기화 되지 않으므로 out이 인 경우에는 1을 더하는 것 대신 직접 out의 값을 으로 초기화 시킨다.) 또한, rst 신호가 없을 때 clk가 posedge이면 out의 값과 상관 없이 out의 값이 으로 초기화 된다. 오른쪽의 test bench 코드는 clk를 10ns마다 보수를 취하고, 400ns마다 rst에 보수를 취함으로써, 4bit decade counter가 가질 수 있는 모든 경우의 수를 고려할 수 있도록 하였고, (4bit decade counter의 모든 상태 개수)이므로 rst의 값이 바뀌기 전에 counter의 모든 상태가 한 번이상 순환될 수 있게 했다. 위 시뮬레이션에서 0~400ns 까지는 rst 신호가 없을 때를 나타내어 out의 값이 으로 유지되는 것(trigger될 때마다 으로 초기화)을 볼 수 있고, 400ns~800ns까지는 rst 신호가 있어 trigger될 때 마다 counter의 값이 ~(BCD Code 0~9)을 순환하는 것을 볼 수 있다.

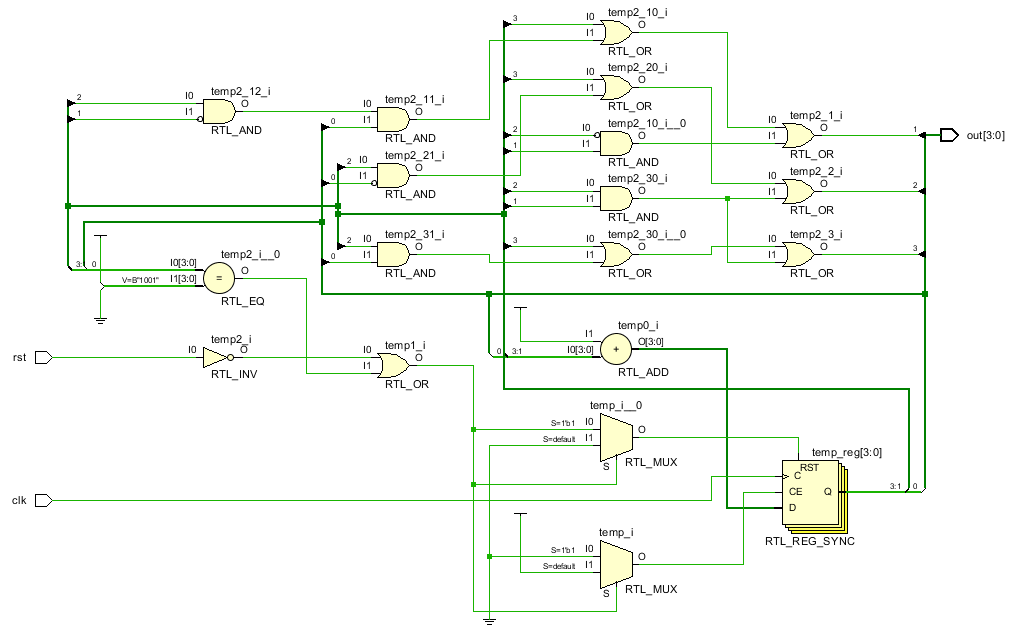
**3.**



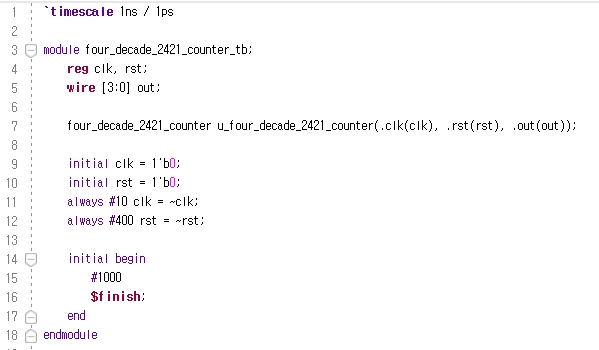
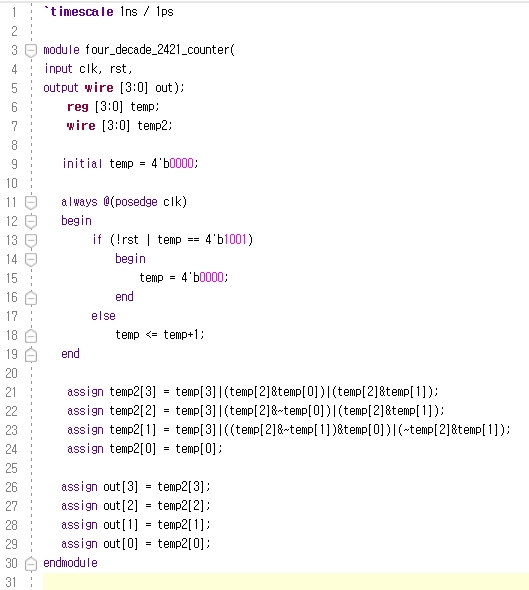
[4bit 2421 decade counter의 Simulation 결과]



[4bit 2421 decade counter의 state table]



[4bit 2421 decade counter의 schematic]



[4bit 2421 decade counter의 Verilog Code]

4bit 2421 decade counter는 4bit 2421 코드를 나타내는 counter로, trigger 될 때마다 , , , , , , , , , 의 상태를 반복하는 회로이다. 4bit 2421 decade counter는 위의 state table과 같이 10개의 상태를 가지고, 생략된 ~의 상태들은 모두 2421 코드에서와 같이 유효하지 않은 값이다.

위 코드에서는 out 대신 temp에 counter의 값을 저장하며(reg 타입, 8421 코드를 저장), temp2를 경유해 out에 2421 코드로 변환된 값을 출력한다. 또한, clk가 posedge(rising edge)일 때 counter가 trigger 된다.

위 코드에서는 4비트의 temp를 먼저 으로 초기화 시킨다. 그 다음, rst(reset)신호가 신호가 없거나(rst=0), temp의 값이 (9)일 때 trigger 되면 temp의 값을 으로 초기화 시킨다. rst 신호가 있을 때(rst=1) trigger 되면 temp의 값에 1을 더해 temp가 다음 상태의 값을 가지게 한다.

그런 다음, temp의 값(8421 코드)을 2421 코드로 변환해 temp2에 저장하고, temp2의 값을 out에 저장한다.

8421 코드(ABCD)에서 2421 코드(A’B’C’D’)로 변환하는 논리식은 6주차 결과 레포트의 8421(BCD)-2421 Code converter에서 카르노맵을 사용하여 구했으며, 다음과 같다.

위 수식을 사용해 위 코드의 21~24번째 줄에서 temp를 2421 코드로 변환한다.

예를 들어 temp의 값이 , , , , , , , , , 일 때 temp2는 , , , , , , , , , 이 되고, temp는 4bit decade counter에서의 out과 동일한 값을 가진다.

결과적으로 out은 4bit 2421 decade counter의 상태(state) 값을 가지게 된다.

또한, rst 신호가 없을 때는 clk가 posedge이면 out의 값과 상관 없이 out의 값이 으로 초기화 된다. 오른쪽의 test bench 코드는 이전과 마찬가지로 clk를 10ns마다 보수를 취하고, 400ns마다 rst에 보수를 취함으로써, 4bit 2421 decade counter가 가질 수 있는 모든 경우의 수를 고려할 수 있도록 하였고, (4bit 2421 decade counter의 모든 상태 개수)이므로 rst의 값이 바뀌기 전에 counter의 모든 상태가 한 번이상 순환될 수 있게 했다. 위 시뮬레이션에서 0~400ns 까지는 rst 신호가 없을 때를 나타내어 out의 값이 으로 유지되는 것(trigger될 때마다 으로 초기화)을 볼 수 있고, 400ns~800ns까지는 rst 신호가 있어 trigger될 때 마다 counter의 값이 ~(2421 Code 0~9, ~제외)을 순환하는 것을 볼 수 있다.

**4.**

이번 실험을 통해 각 binary 카운터, decade 카운터, 2421 decade 카운터의 원리와 작동방식에 대해 알 수 있었다. 실험에서 구현한 각각의 카운터에서 clk가 posedge(rising edge)일 때 trigger되어 state에 변화가 있게 만들었다.

rst가 0일 때는 trigger될 때 state를 초기화() 시키고, rst가 1일 때에는 trigger 될 때 state를 다음 state로 넘어가는 방식으로 작동하는 것을 알 수 있었다.

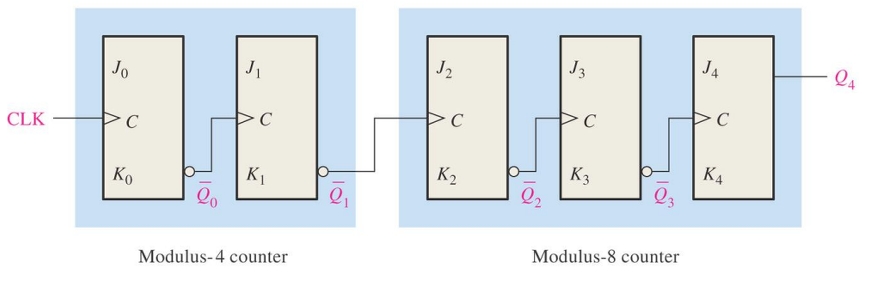
또한, 코드에는 생략되었지만, 각각의 카운터에서는 입력 x와 출력 z를 가질 수 있는데, 입력 x가 0일때는 state가 변하지 않고, x가 1인 경우에는 다음 상태(, 이 마지막 상태면 )로 넘어간다. 또한, 출력 z는 x가 1이고 현재 state가 마지막 state이여서 next state가 처음의 state()가 되는 경우에만 1이 되고, 다른 모든 경우에서는 0이 된다.

이는 각각의 항목의 state table에서 확인 할 수 있다.

마지막으로, 2421 code counter의 경우, out에 직접 2421 코드의 덧셈연산(1증가)을 사용해 구현할 수는 있으나, 너무 코드가 복잡해지므로 이미 구현했던 8421(BCD)-2421 Code converter의 논리식을 활용하여 임시 배열을 통해 8421 code값을 저장하고, 2421 code로 변환하여 out으로 내보내는 방식으로 구현하였다.

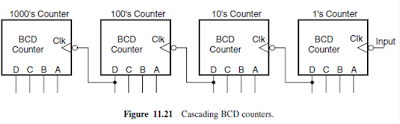
**5.**

- Cascading Counter



[비동기 Counter의 종속 연결 : modulus-32 counter를 modulus-4, modulus-8 카운터로 제작함]

카운터도 가산기처럼 여러 개의 서로 다른 카운터들을 연결함으로써 더 큰 카운터를 만들 수 있다. 예를 들어, 여러 자리수의 10진수 카운터를 만들 때, 아래처럼 BCD 카운터 4개를 연결함으로써 만들 수 있다. (이전자리 MSB를 다음 자리의 clk로 연결함)



처음에는 모든 BCD 카운터가 (0)의 state로 시작하는데, 이전 자리수의 BCD 카운터가 (9)의 상태에서 next state인 (0)로 넘어갈 때, 이전자리의 MSB의 비트가 1에서 0으로 바뀌므로 이 이전자리의 MSB의 falling edge가 다음 자리의 clk에 rising edge로 입력되기 때문에

결과적으로 이전자리가 모든 state를 거쳐 한 바퀴를 돌 때, 다음자리의 state가 1번 변하게 되므로, 위 그림과 같이 4개의 BCD 카운터를 연결한 경우 왼쪽부터 순서대로 , , , 번의 input(가장 낮은 자리수의 BCD 카운터의 clk에 들어오는 신호)의 rising edge마다 state가 변하게 된다. 예를들어, 7364번의 input의 rising edge가 들어왔다면, 맨 앞자리수의 BCD 카운터 부터 각각 , , , 번 next state로 넘어갔으므로 최종 state는 각각 (7), (3), (6), (4)가 되어 7364가 카운트 된 것을 알 수 있다.