12주차 결과보고서

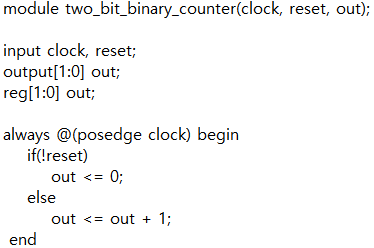
전공: 컴퓨터공학과 학년: 3학년 학번: 20191599 이름: 송경호

**1.**

2-bit 카운터는 0부터 3까지, 총 4개의 상태가 순환하며 변하는 카운터이다. 2 bit의 출력을 가지며, 00 > 01 > 10 > 11의 순으로 상태가 변화하게 된다. 또한 11이후로 다시 0으로 초기화된다.

카운터의 상태 변화는 clock의 변화에 따라 발생한다. clock의 상승 엣지, 혹은 하강 엣지에서 신호의 발생을 통해 상태가 변화하는 원리이다. 또한 rst 비트의 입력을 받아 상태의 초기화가 이루어질 수 있다.

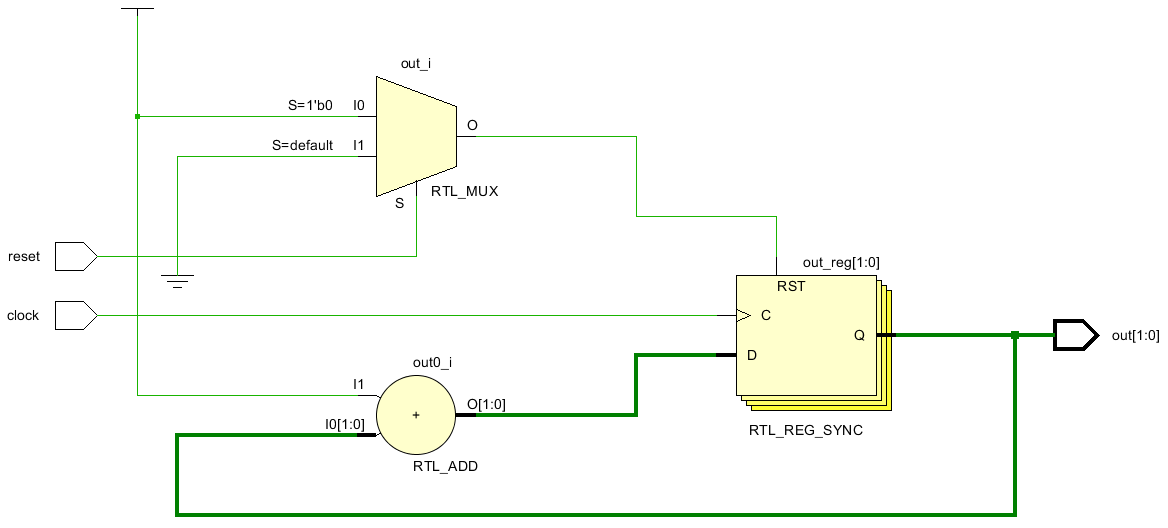
결론적으로 2-bit 카운터는 clock과 reset 두 개의 입력을 받고, 2개의 출력을 발생시키는 회로이다. 다음은 구현한 Verilog 코드이다.

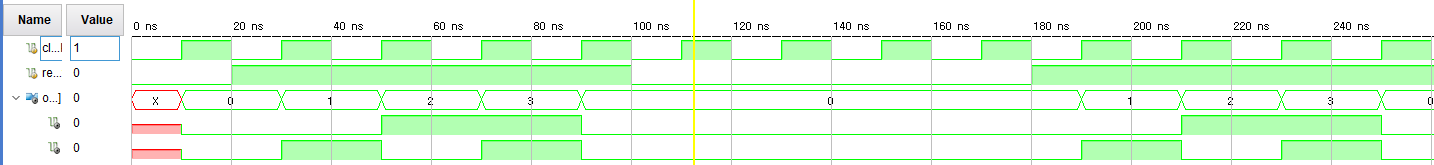
 텍스트, 폰트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명

좌측이 디자인 코드, 우측이 테스트 밴치 코드이다. clock과 reset을 입력 받고 길이 2의 배열을 출력하는 것을 알 수 있다. 또한 앞서 말했듯이, clock의 엣지마다 변화가 발생해야 하므로, *`always @(posedge clock)`* 구문을 통해 clock의 상승 엣지에서 변화가 발생하도록 구현했다. reset 비트가 0 인 경우 출력은 0으로 초기화되며 비트가 1일 경우 출력은 1 씩 증가한다. 따라서 앞서 말한 00 > 01 > 10 > 11의 순으로 출력이 발생한다.

다음은 Schematic과 simulation 결과이다.





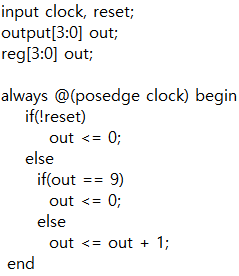
clock은 테스트 밴치 코드에 따라 10ns마다 1과 0을 오간다. 이때 reset bit가 1인 20ns ~ 100ns의 구간을 확인하면 각 각 30, 50, 70, 90 ns 마다 상승 엣지가 발생하는 것을 확인할 수 있다. 따라서 해당 시점에서 출력 비트가 1씩 증가해야 하며, 실제로 30ns에서 00 > 01, 50ns에서 01 > 10, 70ns에서 10 > 11, 90ns에서 11 > 00으로 값이 변화하는 것을 확인할 수 있다. 이후 reset bit가 0일 경우 clock의 상승 엣지가 발생하더라도 출력은 변하지 않는다.

**2.**

4-bit decade 카운터는 기본적으로 2bit 카운터와 동일한 역할을 수행한다. 이때 4 bit 카운터이므로 총 출력 비트가 2개에서 4개로 늘어난다. 이때, 주의할 점은 4 bit로 표현가능한 모든 상태를 출력하는 것이 아니라는 점이다. 즉, 0000부터 1111까지 총 16개의 상태를 출력하는 것이 아닌 1001의 상태이후 다시 0000의 상태로 순환하는 구조로 총 0부터 9까지 10개의 상태를 가지며, 이 점이 decade 카운터라고 불리는 이유이다.

2 bit카운터와 동일하게 상태 변화는 clock의 변화에 따라 발생한다. clock의 상승 엣지, 혹은 하강 엣지에서 신호의 발생을 통해 상태가 변화하고 rst 비트의 입력을 받아 상태의 초기화가 이루어질 수 있다.

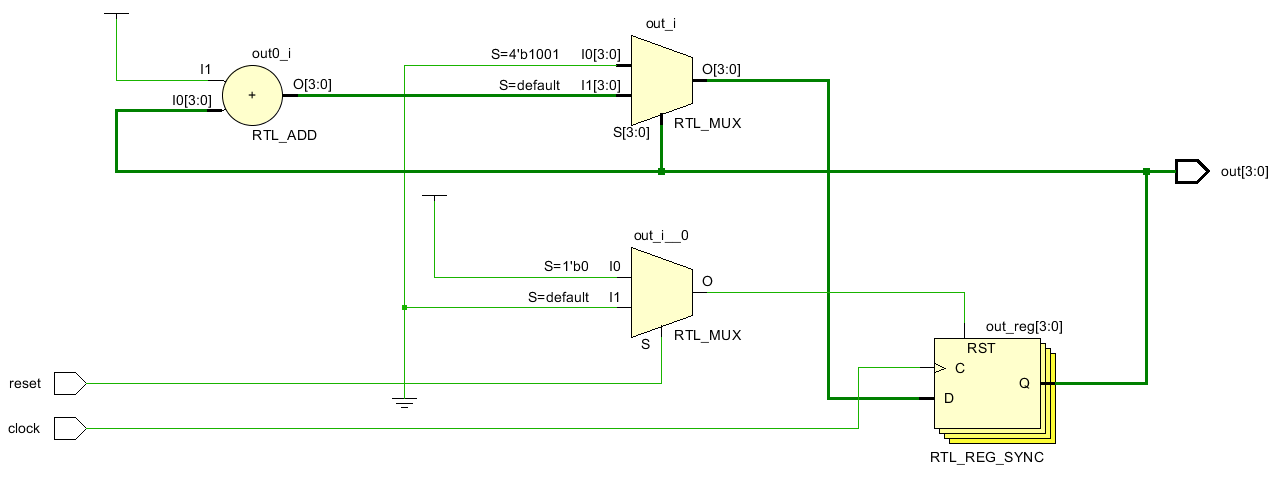
결론적으로 4-bit decade 카운터는 clock과 reset 두 개의 입력을 받고, 4개의 출력을 발생시키며 결과적으로 0000의 출력부터 1001까지의 출력이 순환되는 구조이다.

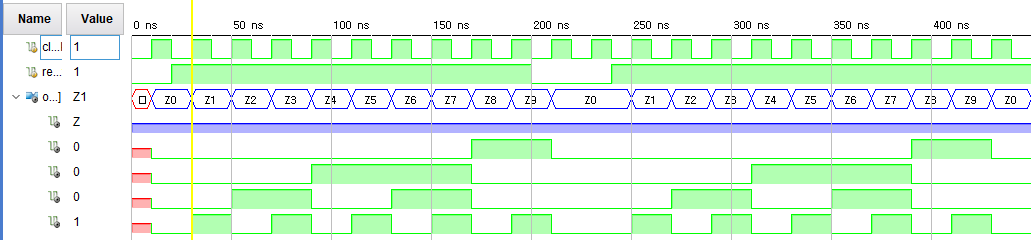
 텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명

좌측이 디자인 코드, 우측이 테스트 밴치 코드이다. clock과 reset을 입력 받고 길이 4의 배열을 출력하는 것을 알 수 있다. 또한 앞서 말했듯이, clock의 엣지마다 변화가 발생해야 하므로, *`always @(posedge clock)`* 구문을 통해 clock의 상승 엣지에서 변화가 발생하도록 구현했다. reset 비트가 0 인 경우 출력은 0으로 초기화되며 비트가 1일 경우 출력은 1 씩 증가한다. 또한 출력이 9(1001)인 경우 다시 출력을 0으로 초기화한다.

다음은 Schematic과 simulation 결과이다.





clock은 테스트 밴치 코드에 따라 10ns마다 1과 0을 오간다. 이때 reset bit가 1인 20ns ~ 200ns의 구간을 확인하면 각 각 30, 50, … ,190ns 마다 상승 엣지가 발생하는 것을 확인할 수 있다. 따라서 해당 시점에서 출력 비트가 1씩 증가해야 하며, 실제로 30ns에서 0000 > 0001, 50ns에서 0001 > 0010, 70ns에서 0010 > 0011 순으로 증가하고 계속해서 이어나가 190ns에서 1000 > 1001로 증가한다. 이후에 다시 reset 비트가 0이 되며 상승 엣지가 발생함에도 출력비트가 00으로 변화가 없는 것을 알 수 있다. 또한 1001이후 다시 0000으로의 출력 변화는 430ns에서 발생한 상승 엣지에서 확인할 수 있다.

**3.**

4-bit 2421 decade 카운터는 위에 설명한 4-bit decade 카운터와 동일하다. 위에서 말한 4-bit decade 카운터는, 정확히는 4-bit 8421 decade counter를 의미하는 것으로 4개의 비트가 각 각 8, 4, 2, 1의 가중치를 갖는다. 따라서 9라는 숫자는 1001을 표현 가능하다. 이와 달리 2421 decade 카운터는 4개의 비트가 각 각 2, 4, 2, 1의 가중치를 갖는다. 때문에, 예를 들어 9라는 숫자를 표현하기 위해선 8421 카운터와 달리 1111이 필요하다. 다음은 2421 카운터의 진리표이다.

|  |
| --- |
| q0 (=0000) |
| q1 (=0001) |
| q2 (=0010) |
| q3 (=0011) |
| q4 (=0100) |
| q5 (=1011) |
| q6 (=1100) |
| q7 (=1101) |
| q8 (=1110) |
| q9 (=1111) |

진리표를 확인하면, 4를 나타내는 0100까지는 8421과 같은 값을 갖지만 5ㅂ터 1011으 출력을 갖고 이후로 다시 1씩 증가한 다는 것을 알 수 있다.

상태 변화는 역시 clock의 변화에 따라 발생한다. clock의 상승 엣지, 혹은 하강 엣지에서 신호의 발생을 통해 상태가 변화하고 rst 비트의 입력을 받아 상태의 초기화가 이루어질 수 있다.

결론적으로 동일하게 clock과 reset 두 개의 입력을 받고, 4개의 출력을 발생시키며 결과적으로 위의 진리표에 따른 순환을 발생시킨다.

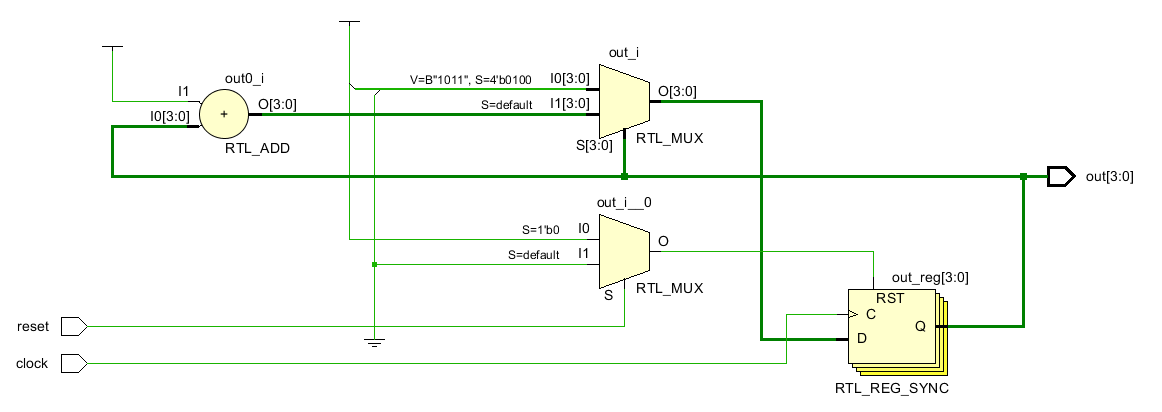
텍스트, 스크린샷, 폰트이(가) 표시된 사진

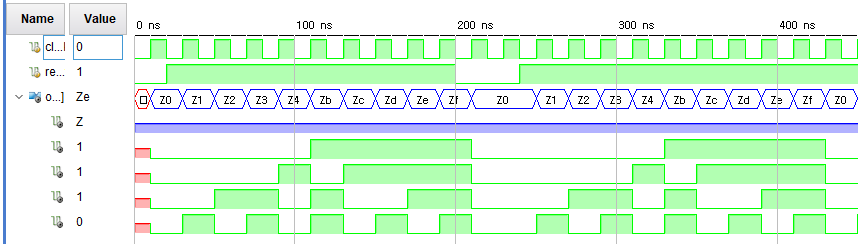
자동 생성된 설명 텍스트, 폰트, 스크린샷, 대수학이(가) 표시된 사진

자동 생성된 설명

좌측이 디자인 코드, 우측이 테스트 밴치 코드이다. clock과 reset을 입력 받고 길이 4의 배열을 출력하는 것을 알 수 있다. 또한 앞서 말했듯이, clock의 엣지마다 변화가 발생해야 하므로, *`always @(posedge clock)`* 구문을 통해 clock의 상승 엣지에서 변화가 발생하도록 구현했다. reset 비트가 0 인 경우 출력은 0으로 초기화되며 비트가 1일 경우 출력은 1 씩 증가한다. 또한 출력이 0100(4)인 경우 2421의 규칙에 따라 다음 출력을 1011로 변화시키며 다시 1씩 증가시킨다. 8421 clock과 달리 9의 표현이 1111이므로 자동적으로 순환되게 된다.

다음은 Schematic과 simulation 결과이다.





clock은 테스트 밴치 코드에 따라 10ns마다 1과 0을 오간다. 이때 reset bit가 1인 20ns ~ 200ns의 구간을 확인하면 각 각 30, 50, … ,190ns 마다 상승 엣지가 발생하는 것을 확인할 수 있다. 따라서 해당 시점에서 출력 비트가 1씩 증가해야 하며, 실제로 30ns에서 0000 > 0001, 50ns에서 0001 > 0010, 70ns에서 0010 > 0011 순으로 증가한다. 이때 110ns 지점에서 출력이 0100에서 1011로 바뀌는 것을 볼 수 있으며 이후로는 계속해서 1씩 증가하여 1111까지 출력이 증가한다는 것을 알 수 있다. 이후에 다시 reset 비트가 0이 되며 상승 엣지가 발생함에도 출력비트가 00으로 변화가 없는 것을 알 수 있다. 또한 1111이후 다시 0000으로의 출력 변화는 430ns에서 발생한 상승 엣지에서 확인할 수 있다.

**4.**

실험 목표인 3개의 카운터 구현이 성공적으로 이루어진 것을 알 수 있었다. Counter의 특성 상 모두 순환되는 출력 형식을 갖고, clock의 상승 또는 하강 엣지에 출력이 변화하며, reset 비트가 활성화되어 있어야 기능이 동작함을 simulation 결과를 통해 확인할 수 있었다. 또한 FPGA 구현 결과 reset에 해당하는 스위치를 누른 채로, clock 스위치를 누를 때마다 출력이 변환되는 것을 확인하며 올바른 구현을 확인할 수 있었다.

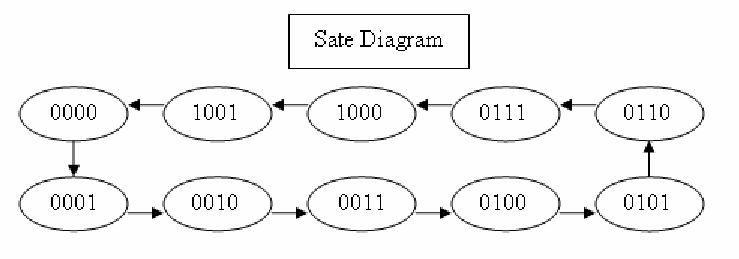
4개의 카운터 구현은 기본적으로 비슷하나, 디자인코드에서 posedge시 발생하는 변화의 규칙을 설정해줌으로 써 구현 가능했다.

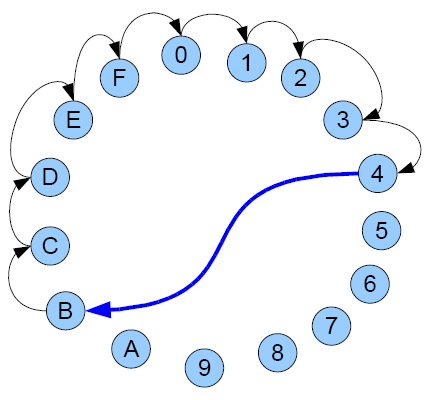
**5.**

**5-1) State Diagram**

3개의 clock을 state diagram을 통해 표현 가능하다.



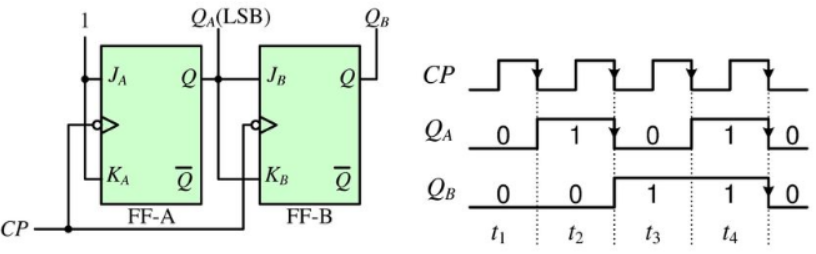




위에서부터 각 각, 2bit, 4bit 8421, 4bit 2421 counter의 state diagram이다. state diagram을 이용하면 보다 직관적으로 clock의 순환 구조를 확인할 수 있다.

**5-2) Synchronous, Asynchronous**

Counter에는 동기식과 비동기식 카운터가 존재한다. 동기식 카운터는 하나의 clock 입력을 카운터를 구성하는 모든 flipflop이 공유하는 방식이며 반대로 비동기식 카운터는 최하위 플립 플롭만이 clock입력을 받고 해당 플립 플롭이 상위 플립 플롭의 출력을 결정하는 방식이다. 이번 실험에서는 모두 비동기식 방식으로 카운터를 구현하였으며 2 bit counter의 동기식 구현은 아래와 같다.



동기식 카운터의 경우 gate delay가 발생하지 않아 비동기식 카운터에 비해 값의 변경이 빠르다는 특징이 있다.