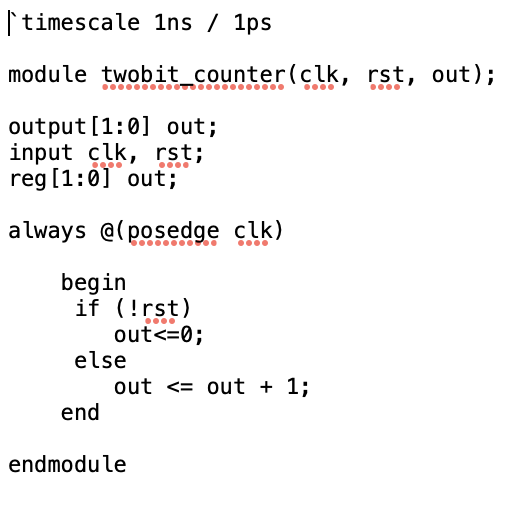
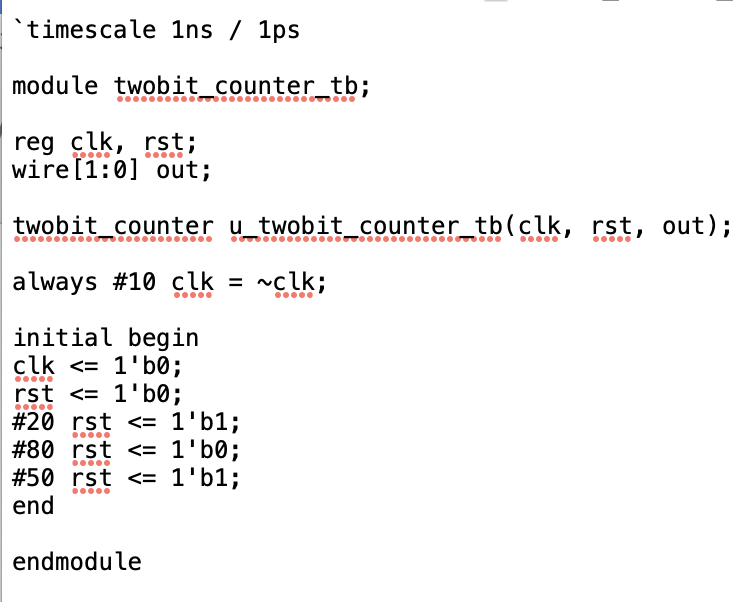
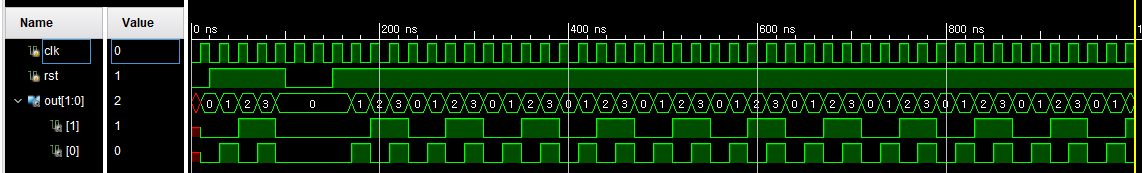
12주차 결과보고서

전공: 컴퓨터공학과/생명과학과 학년: 9학기 학번: 20181435 이름: 박다희

**1.**

****

****

**도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

Clock pulse가 있을 때마다 00, 01, 10, 11 순서를 반복하는 2-bit counter을 구현하기 위해 clk가 상승 edge일 때 rst가 0이 아니라면 out 값은 1이 증가하고, rst가 0이라면 out 값이 0이 되도록 설정하고 counter을 초기화해주는 값으로 값이 0일 경우의 rst를 사용했으며 00, 01, 10 , 11 순으로 출력 값이 하나씩 커지는 것을 구현하기 위해 out <= out + 1로 구현하였다. Simulation의 결과를 보면 rst가 0일 때는 out[0]과 out[1]이 둘 다 0으로 초기화되는 것을 볼 수 있고, rst가 1일 때 out[0] =0, out[1] = 0부터 시작해서 out[0] =1, out[1] = 0 그리고 out[0] =0, out[1] = 1 그리고 out[0] =1, out[1] = 1 순으로 값을 갖는 것을 확인할 수 있다. 이에 대한 schematic diagram을 보면 RTL\_MUX을 통해 입력들 중 하나를 선택하여 출력으로 전달하고 있고, RTL\_ADD는 코드에서 out <= out + 1의 덧셈 논리를 구현하고 있다. 또한 RTL\_REG\_SYNC은 현재 count 값을 저장하고 clock 신호에 따라 이를 업데이트 하여 다음 count 값을 만드는 데 사용된다.

**2.**

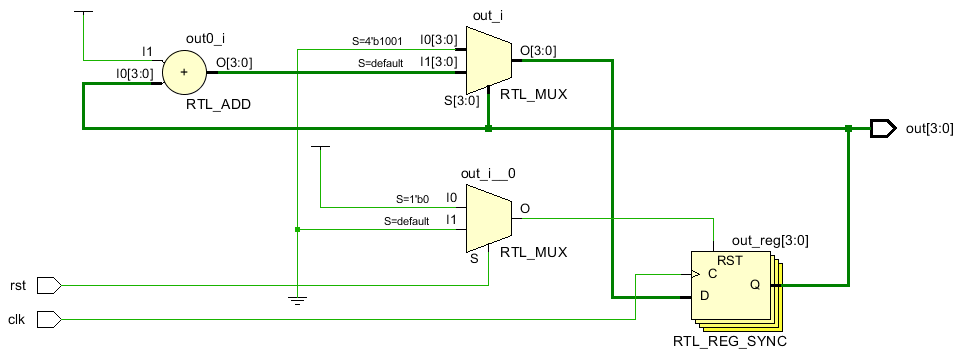
**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**스크린샷이(가) 표시된 사진

자동 생성된 설명**

****

4-bit decade counter는 4-bit binary counter와는 다르게 4-bit 이진수로 표현할 수 있는 16개의 상태 말고 0에서 9까지의 상태만을 나타내는 counter이다. 따라서 0000에서 1001까지만을 반복하는데, clock pulse가 나타날 때마다 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001 순으로 반복하는 4-bit decade counter을 구현하기 위해 clock이 상승 edge이고 rst가 1일 때 out이 9가 아니라면 out <= out + 1로 값을 하나씩 증가시키고, out이 9라면 out 값을 0으로 설정해주었다. 또한 초기화로 rst가 0이라면 out 값을 0으로 초기화하였다. 이러한 Verilog 코드를 바탕으로 한 simulation 결과를 살펴보면 다음과 같다. Clock이 상승 edge일 때 rst의 값이 0이라면 out[3], out[2], out[1], out[0]의 값이 모두 0으로 초기화 되는 것을 확인할 수 있고, clock이 상승 edge일 때 rst의 값이 1이라면 out[3]=0, out[2]=0, out[1]=0, out[0]=1 그리고 out[3]=0, out[2]=0, out[1]=1, out[0]=0 그리고 out[3]=0, out[2]=0, out[1]=1, out[0]=1 순으로 값이 변하고 있음을 볼 수 있다. 이렇게 1씩 계속 커지다가 십진수로 9인 out[3]=1, out[2]=0, out[1]=0, out[0]=1이 되면 그 다음 값은 out[3]=0, out[2]=0, out[1]=0, out[0]=0로 십진수 0으로 초기화 됨도 확인할 수 있다. 이를 바탕으로 한 schematic diagram을 보면 앞서 2-bit counter의 schematic diagram에서 본 것과 같이 RTL\_ADD, RTL\_MUX, RTL\_REG\_SYNC 로 이루어져 있는 것을 볼 수 있다. 차이점은 RTL\_MUX가 2개 쓰인다는 점인데 2개의 RTL\_MUX는 여러 입력 신호들 중에서 하나를 선택하고 RTL\_ADD는 out이 9가 아닐 때 out <= out + 1의 덧셈 연산을 수행한다. 마지막으로 RTL\_REG\_SYNC은 현재 count 값을 저장하고 clock 신호에 따라 이를 업데이트 하여 다음 count 값을 만드는 데 사용된다.

**3.**

**텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명**

**스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명**

**도표, 라인, 텍스트, 평면도이(가) 표시된 사진

자동 생성된 설명**

4-bit 2421 decade counter는 0부터 4까지는 0000, 0001, 0010, 0011, 0100으로 4-bit decade counter와 같다가 5부터는 1011, 6은 1100, 7은 1101, 8은 1110, 9는 1111로 나타내는 counter이다. 따라서 0000에서 1111까지의 16상태 중에서 0101, 0110, 0111, 1000, 1001, 1010을 제외한 다른 상태를 나타내는 counter인데, clock pulse가 나타날 때마다 0000, 0001, 0010, 0011, 0100, 1011, 1100, 1101, 1110, 1111 순으로 반복하는 4-bit 2421 decade counter을 구현하기 위해 clock이 상승 edge이고 rst가 1일 때 out이 4와 15가 아닐 때는 out <= out + 1로 값을 하나씩 증가시키고, out이 4라면 out 값을 보수 취하고, out이 15일 때는 0으로 설정해주었다. 또한 초기화로 rst가 0이라면 out 값을 0으로 초기화하였다. 이러한 Verilog 코드를 바탕으로 한 simulation 결과를 살펴보면 다음과 같다. Clock이 상승 edge일 때 rst의 값이 0이라면 out[3], out[2], out[1], out[0]의 값이 모두 0으로 초기화 되는 것을 확인할 수 있고, clock이 상승 edge일 때 rst의 값이 1이라면 out[3]=0, out[2]=0, out[1]=0, out[0]=1 그리고 out[3]=0, out[2]=0, out[1]=1, out[0]=0 그리고 out[3]=0, out[2]=0, out[1]=1, out[0]=1 순으로 값이 변하고 있음을 볼 수 있다. 이렇게 1씩 계속 커지다가 십진수로 4인 out[3]=0, out[2]=1, out[1]=0, out[0]=0이 되면 그 다음 값은 4의 보수인 out[3]=1, out[2]=0, out[1]=1, out[0]=1이 나타나고 있음을 볼 수 있다. 그리고 그 이후부터 9를 나타내는 out[3]=1, out[2]=1, out[1]=1, out[0]=1까지 값이 1씩 증가함을 확인할 수 있고, 9 다음은 out[3]=0, out[2]=0, out[1]=0, out[0]=0인 십진수 0이 되는 것도 확인할 수 있다. 이를 바탕으로 한 schematic diagram을 보면 앞서 4-bit decade counter의 schematic diagram에서 본 것과 같이 RTL\_ADD, RTL\_MUX, RTL\_REG\_SYNC 로 이루어져 있는 것을 볼 수 있다. 차이점은 RTL\_MUX가 2개 쓰인 4-bit decade counter와는 다르게 4-bit 2421 decade counter은 RTL\_MUX가 3개 쓰인다는 점인데 3개의 RTL\_MUX는 여러 입력 신호들 중에서 하나를 선택하고 RTL\_ADD는 out이 4 또는 15가 아닐 때 out <= out + 1의 덧셈 연산을 수행한다. 마지막으로 RTL\_REG\_SYNC은 현재 count 값을 저장하고 clock 신호에 따라 이를 업데이트 하여 다음 count 값을 만드는 데 사용된다.

**4.**

이번 주차 실험에서는 2-bit counter, 4-bit decade counter 그리고 4-bit 2421 decade counter을 구현하고 simulation결과와 schematic diagram까지 확인해 보았다. Counter는 출력 값인 out을 0로 초기화해주지 않으면 임의의 값에서부터 count가 시작되기 때문에 리셋 입력 값인 rst를 사용하였고, 각각의 counter을 구현할 때는 2-bit counter는 rst의 값만 확인하며 out 값을 1씩 증가시켰지만, 4-bit decade counter와 4-bit 2421 decade counter는 out 값이 9이거나 15이거나 4일 때 단순히 1을 증가시키는 것이 아닌 다른 값으로 지정해줘야 하므로 if, else if, else 문으로 상황들을 나누어 구현해주었다. FPGA 동작 결과와 simulation, schematic diagram을 살펴본 결과 2-bit counter, 4-bit decade counter 그리고 4-bit 2421 decade counter 모두 잘 동작함을 확인할 수 있었다.

**5.**

이번 주차에 구현한 counter들을 이용하여 여러 다양한 진수의 카운터들을 나타낼 수 있는데, 이를 구현하기 위해서는 flip-flop이 몇 개 필요한 지를 알아야 한다. 보통 진 카운터를 구현할 때 n개의 flip-flop을 사용한다. 예를 들면 3진 카운터는 2개의 flip-flop을, 6진 카운터는 3개의 flip-flop을, 10진 카운터는 4개의 flip-flop을 필요로 한다. 밑의 표는 6진 카운터의 상태표이다.

텍스트, 번호, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위의 표를 보면 clock pulse가 나타날 때마다 0부터 5까지의 출력 값을 순서대로 갖는다.