**12주차 결과보고서**

**전공: 경영학과 학년: 3학년 학번: 20190963 이름: 한다현**

**1.**

**2bit counter는 00, 01, 10, 11의 상태를 갖는 4개의 수를 셀 수 있는 카운터이다. 실험에서 설계한 2bit counter는 clock pulse가 0에서 1로 상승하는 positive edge에서 동작하며, reset 입력이 1이 되면 처음 상태로 돌아간다. 따라서 이 카운터의 입력은 clock pulse를 의미하는 Clk와 reset이 있고, 카운터의 출력은 2bit 이진수인 out이 있다. 이 때 out은 배열로 선언한다. Clk와 reset, 그리고 out을 모듈에서 선언하는 Verilog 코드는 다음과 같다.**

**module two(Clk, rst, out);**

**input Clk, rst;**

**output[1:0]out;**

**reg[1:0]out;**

**이 때 reg[1:0]out은 다음 상태가 할당되기 전까지 기존의 상태를 저장하기 위한 코드이다. 입력과 출력을 선언한 후에 본격적으로 카운터를 설계하는 코드는 다음과 같다.**

**initial out = 2'b00;**

**always @(posedge Clk)**

**begin**

**if (rst == 1'b1)**

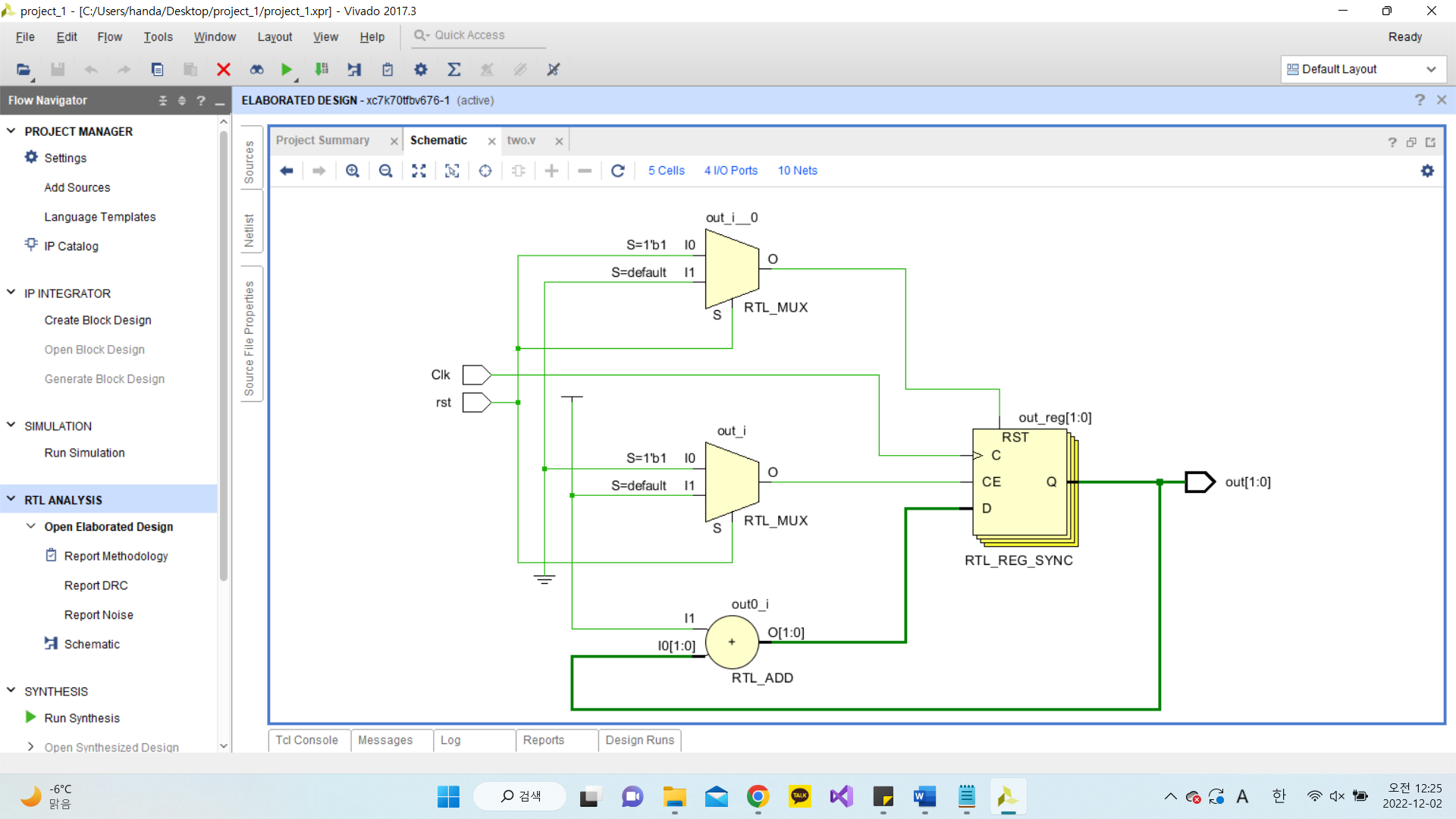
**out = 2'b00;**

**else**

**out <= out + 1;**

**end**

**우선 initial out = 2’b00; 코드를 통해 출력 out의 초기 값을 00으로 설정한다. 이 후 always문을 시작하는데, poseedge Clk라는 코드는 Clk가 0에서 1로 상승하는 에지일 때 always문 내에 있는 코드를 실행한다는 뜻으로, 이 카운터가 positive edge에서 트리거 되는 카운터라는 것을 의미한다. Reset을 뜻하는 rst가 1이면, out의 값이 다시 초기 상태로 돌아가고, rst가 1이 아닌 0이라면, out은 out에 1을 더한 값이 할당된다. 즉, 상태가 00이고, rst가 0이라면, out은 01이 되는 것이다. 이 후 디자인 코드의 종료를 의미하는 endmodule 코드로 디자인 코드를 종료한다. 이렇게 설계한 디자인 코드로 schematic을 확인하면 다음과 같다.**

****

**이제 입력의 변화에 따라 출력이 변화하는 것을 확인하기 위해 시뮬레이션 코드를 작성해야 하는데, 전체적인 코드는 다음과 같다.**

**module two\_tb;**

**reg Clk, rst;**

**wire[1:0]out;**

**two two\_tb(**

**.Clk(Clk),**

**.rst(rst),**

**.out(out));**

**initial begin**

**Clk = 1'b0;**

**rst = 1'b0;**

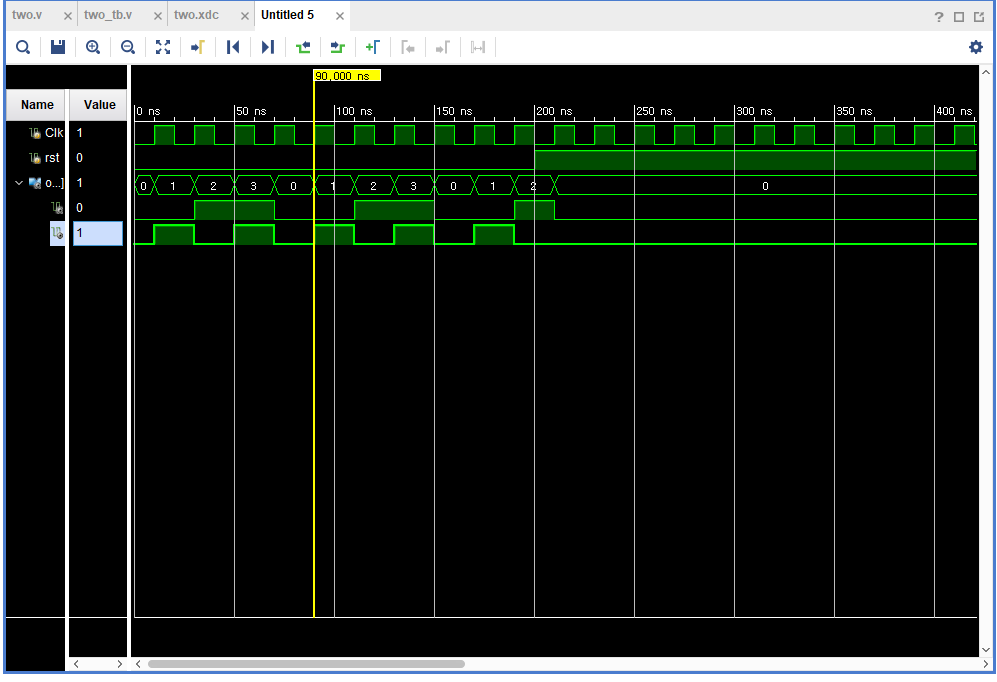
**end**

**always #10 Clk = ~Clk;**

**initial rst <= #200 1'b1;**

**endmodule**

**먼저 입력 Clk, rst와 출력 out을 시뮬레이션 코드에 연결시켜준다. 이 후에 Clk의 초기 값을 0으로, rst의 초기 값도 0으로 설정한 뒤, Clk는 일정한 간격에 따라 반전되도록 설계하고, rst는 일정 시간 뒤에 1로 변화하도록 설정한다. 이 코드로 시뮬레이션을 실행하면 그 결과는 다음과 같다.**

****

**위 시뮬레이션에서 보여지는 출력 예시는 Clk가 0에서 1로 상승하는 시점이고, 이 시점에 rst의 값은 0이며, 이전의 카운터의 상태는 00이다. 카운터의 상태가 00일 때, Clk가 0에서 1로 상승하는 positive edge이기 때문에 카운터가 동작하며, rst가 0이기 때문에 상태가 초기화 되지 않는다. 따라서 상태는 00에서 1이 증가한 01이 되어야 하는데, 출력 out의 값이 01이므로, 정상적으로 출력되었다고 할 수 있다. 상태가 01로 변화한 후에, 또다시 Clk가 0에서 1로 상승하는 positive edge에서 rst의 값이 0이기 때문에 01에 1이 더해져 상태가 10이 되고, 이는 십진수로 2를 의미하기 때문에 시뮬레이션 결과가 올바르게 출력된 것을 볼 수 있다. 또한 200ns 이전에는 rst의 값이 0이기 때문에 카운터가 계속 동작하지만 시뮬레이션에서 설정한 200ns가 지나는 시점부터는 rst의 값이 1이 되어 카운터의 상태가 초기화된다. 따라서 입력 rst도 정상적으로 동작한다는 것을 알 수 있다.**

**2.**

**4bit decade counter는 0부터 9까지 셀 수 있는 카운터로 4개의 플립플롭으로 구성되어 있다. 4개의 플립플롭으로 만들 수 있는 입력 조합은 0000부터 1111까지 총 16개이지만 decade counter에서는 0부터 9까지 10개의 수만 세는 기능을 하기 때문에 0000부터 1001까지 상태가 차례로 변화하고, 1001 후에는 다시 0000으로 돌아가서 0부터 다시 카운트한다. 이 카운트에서도 2 bit counter와 동일하게 Clk와 rst 입력이 필요하다. Clk는 clock pulse를 의미하고, rst는 reset을 의미한다. 카운터의 출력은 2 bit counter와 다르게 4비트 크기의 출력이 필요하다. 모듈에서 입력 Clk, rst와 출력 out을 선언하는 코드는 다음과 같다.**

**module decade(Clk, rst, out);**

**input Clk, rst;**

**output[3:0]out;**

**reg[3:0]out;**

**여기서도 마찬가지로 reg[3:0]out; 코드가 필요한데, 이는 다음 상태가 할당되기 전까지 기존의 상태를 저장하기 위한 코드이다. 모듈에서 입력과 출력을 선언한 후에 본격적으로 4bit decade counter를 설계하는 코드는 다음과 같다.**

**initial out = 4'b0000;**

**always @(posedge Clk)**

**begin**

**if (rst == 1'b1)**

**out = 4'b0000;**

**else**

**if (out == 4'b1001)**

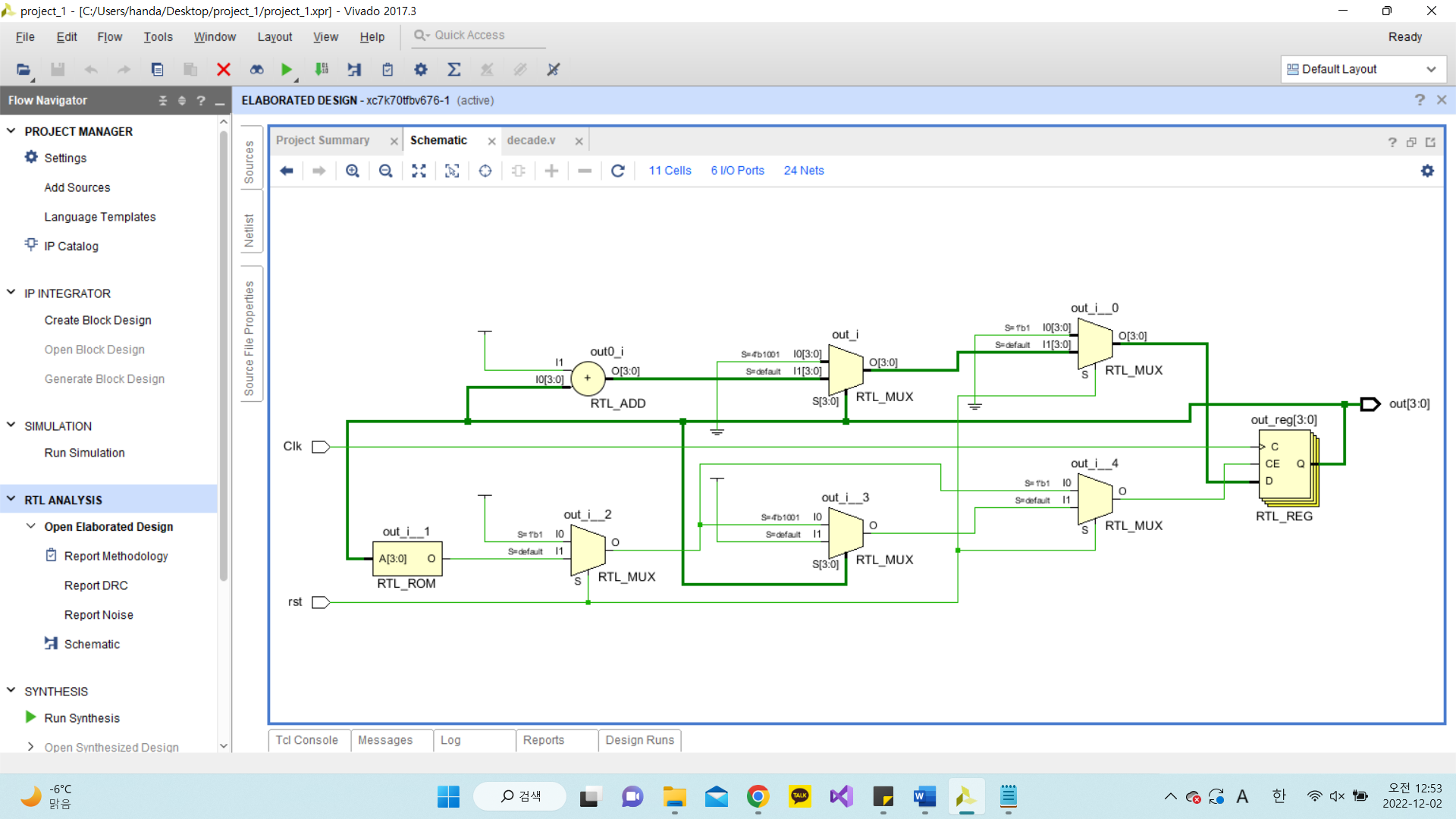
**out = 4'b0000;**

**else**

**out <= out + 1;**

**end**

**먼저 처음 out의 값을 0000으로 초기화 해준다. 이 후 always문이 시작되는데, 2 bit counter와 마찬가지로, Clk가 0에서 1로 상승할 때 카운터가 실행되도록 positive edge를 뜻하는 posedge Clk를 작성한다. 상승 에지에서 always 내에 있는 코드가 실행되는데, 먼저 rst의 값이 1이 되면 out의 값이 다시 0000으로 돌아가 초기화되고, 처음부터 다시 카운트를 시작한다. Rst의 값이 0이면, 다시 두 가지 상황으로 구분된다. Out의 값이 1001, 즉 4 bit decade counter의 마지막 상태인 9가 된다면, 그 다음 out의 상태는 처음 상태인 0000으로 돌아가 다시 처음부터 카운트를 시작한다. 이 코드를 통해 decade counter가 1001보다 큰 상태로 변화하지 않고 카운터의 상태가 초기 값으로 돌아가도록 설계할 수 있다. Rst의 값이 0이고, out의 값이 1001이 아닌 경우에는 out의 값에 1을 더해준다. 예를 들어 십진수 3을 의미하는 0011에서 1을 더해주면 다음 상태는 십진수 4를 의미하는 0100이 된다는 것이다. 4bit decade counter 설계를 마치기 위해 endmodule로 디자인 코드를 종료한다. 이렇게 설계한 디자인 코드를 Verilog에서 schematic으로 확인하면 다음과 같다.**

****

**이제 입력의 변화에 따라 출력이 변화하는 것을 확인하기 위해 시뮬레이션 코드를 작성해야 하는데, 전체적인 코드는 다음과 같다.**

**module decade\_tb;**

**reg Clk, rst;**

**wire[3:0]out;**

**decade decade\_tb(**

**.Clk(Clk),**

**.rst(rst),**

**.out(out));**

**initial begin**

**Clk = 1'b0;**

**rst = 1'b0;**

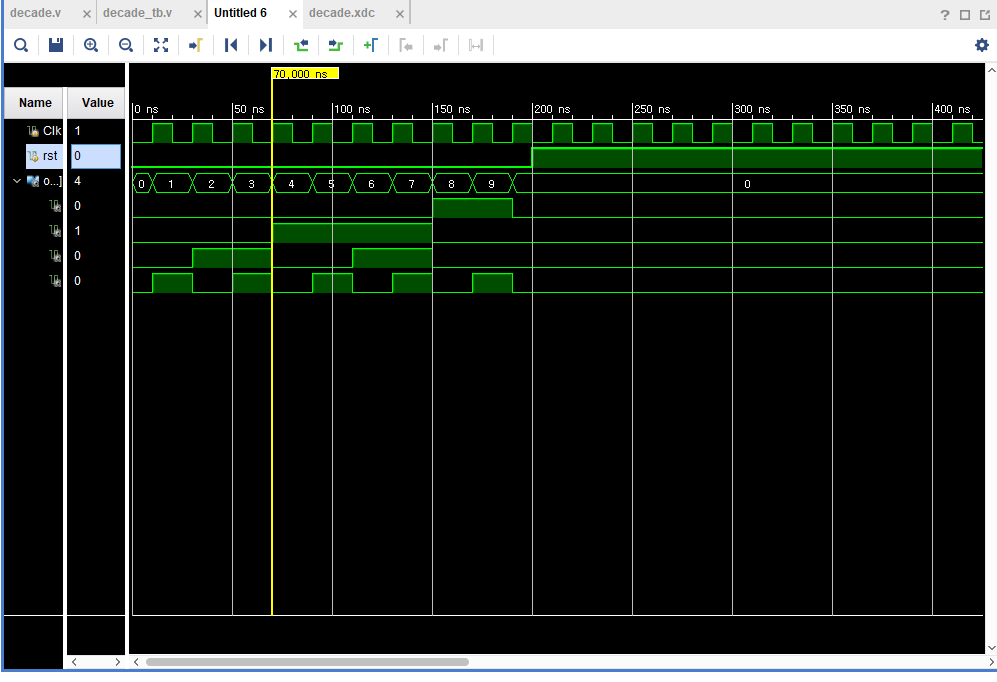
**end**

**always #10 Clk = ~Clk;**

**initial rst <= #200 1'b1;**

**endmodule**

**4 bit decade counter의 시뮬레이션 코드는 2 bit counter의 시뮬레이션 코드와 유사하다. 차이점은 역시 출력 out이 2비트 크기의 배열이 아닌 4비트 크기의 배열이라는 것이다. 먼저, 입력 Clk, rst와 출력 out을 시뮬레이션 코드에 연결시킨다. 그 후에 Clk의 초기 값을 0으로, rst의 초기 값을 0으로 설정한 뒤, Clk는 일정 간격에 따라 반전되도록 설계하고, rst의 값은 일정 시간이 지난 후에 1로 변화하도록 설계한다. 이 시뮬레이션에서는 2 bit counter 시뮬레이션과 동일하게 Clk는 10ns마다 반전되고, rst는 200ns에서 1로 변화하도록 설정하였다. 이 코드를 실행한 시뮬레이션 결과는 다음과 같다.**

****

**위 시뮬레이션 결과에서 보여지는 실행 예시는 Clk가 0에서 1로 상승하는 시점이고, 이 때 rst의 값은 0이며, 카운터의 이전 상태는 0011, 십진수로 3이다. Clk가 0에서 1로 상승하는 시점이기 때문에 positive edge이므로 카운터가 트리거 된다. 그리고 이 때 rst의 값이 0이기 때문에 카운터가 초기화 되지 않고 out에 1이 더해진다. 이전 상태는 십진수로 3을 의미하는 0011이었기 때문에 여기에 1을 더하면 out은 0100, 십진수로 4가 출력된다. 상태가 0100으로 변화한 뒤에 또다시 Clk가 0에서 1로 상승하는 positive edge 시점을 보면, 이 때도 rst가 0이므로 카운터의 상태가 초기화 되지 않고 out에 1이 더해진다. 그러면 0100이었던 상태가 0101로 변화하고 이는 십진수로 5를 의미하기 때문에 시뮬레이션을 확인해봤을 때 정상적으로 작동했다는 것을 알 수 있다. 또한 200ns를 지나는 시점에 rst의 값이 1로 변화한다. Rst의 값이 1로 변화하면 카운터의 상태를 0000으로 초기화하기 때문에 200ns 이후에는 카운터의 상태가 항상 0000이 된다.**

**3.**

**다음으로는 4bit 2421 decade counter를 설계하였다. 4bit 2421 decade counter도 10진수를 카운트하는 장치이다. 앞서 설계한 4 bit decade counter와의 차이점은 2421 코드를 사용한다는 것이다. 2421 코드는 십진수 0에서 9까지를 나타내는 코드인데, 십진수 0부터 9까지를 의미하는 2421 코드는 각각 0000, 0001, 0010, 0011, 0100, 1011, 1100, 1101, 1110, 1111이다. 0000부터 1001까지 증가하는 8421 코드와 차이가 있다. 따라서 4 bit 2421 decade counter를 설계하기 위해서는 8421 코드를 2421 코드로 변환하는 과정이 필요하다. 우선 기존 카운터를 설계하는 것과 동일하게 입력과 출력을 선언해야 한다. 이 카운터에도 Clk와 rst가 입력되며 out이 출력된다. 이 때 out은 4비트 크기이기 때문에 입력과 출력은 다음과 같이 선언된다.**

**module decade\_2421(Clk, rst, out);**

**input Clk, rst;**

**output[3:0]out;**

**reg[3:0]out2;**

**4 bit decade counter와의 차이점은 reg[3:0]out2; 이다. 4 bit decade counter는 코드를 변환하는 과정이 필요하지 않았지만 4 bit 2421 decade counter는 8421 코드를 2421 코드로 변환하는 과정이 필요하기 때문에 출력과 다른 4비트 크기의 out2가 필요하다.**

**initial out2 = 4'b0000;**

**always @(posedge Clk)**

**begin**

**if (rst == 1'b1 | out2 == 4'b1001)**

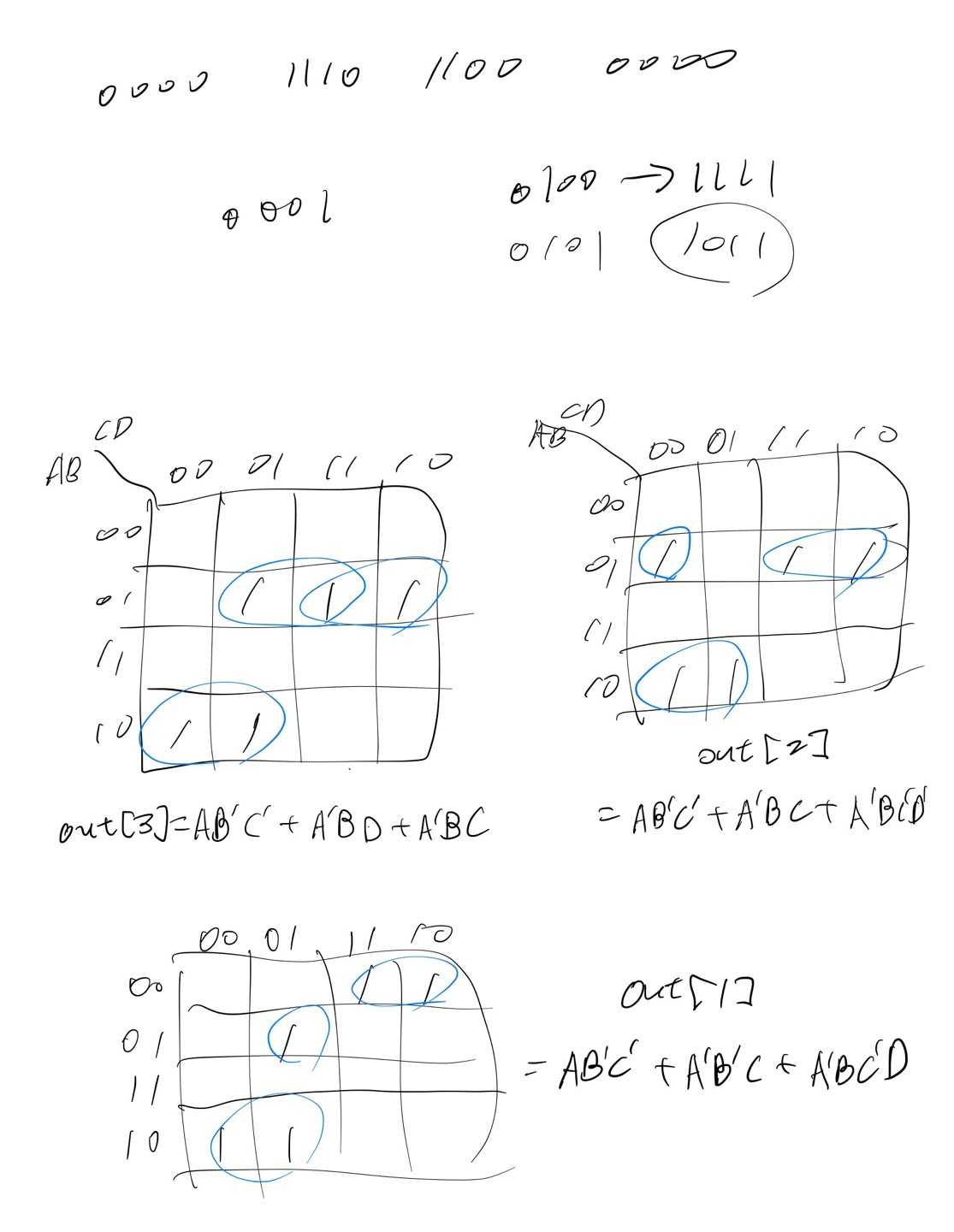
**out2 = 4'b0000;**

**else**

**out2 <= out2 + 1;**

**end**

**우선 8421 코드를 저장하는 out2의 초기 값을 0000으로 설정한다. 이 후 positive edge를 의미하는 posedge Clk를 만족하면 always문이 실행된다. Positive edge 일 때, rst가 1이거나 out2의 값이 십진수 9를 의미하는 1001인 경우에 다시 초기 상태로 돌아가야 하기 때문에 out2의 값이 0000이 된다. Rst의 값이 0이고, out2의 값이 1001이 아닐 때는 out2의 값에 1을 더해준다. Out2는 8421 코드를 나타내기 때문에 이를 2421 코드로 변환하는 코드가 필요하다. Out2의 각각의 비트를 사용하여 out의 각각의 비트를 정의해야 하는데 out[0]은 out2[0]과 항상 같은 값을 갖기 때문에 out[0] = out2[0]이고, out[3] ~ out[1]의 카르노 맵은 다음과 같다.**

****

**A ~ D는 각각 out2[3] ~ out2[0]을 의미하고, 이를 바탕으로 out[3] ~ out[1]을 정의할 수 있다. Out[3] ~ out[0]에 대한 코드는 다음과 같다.**

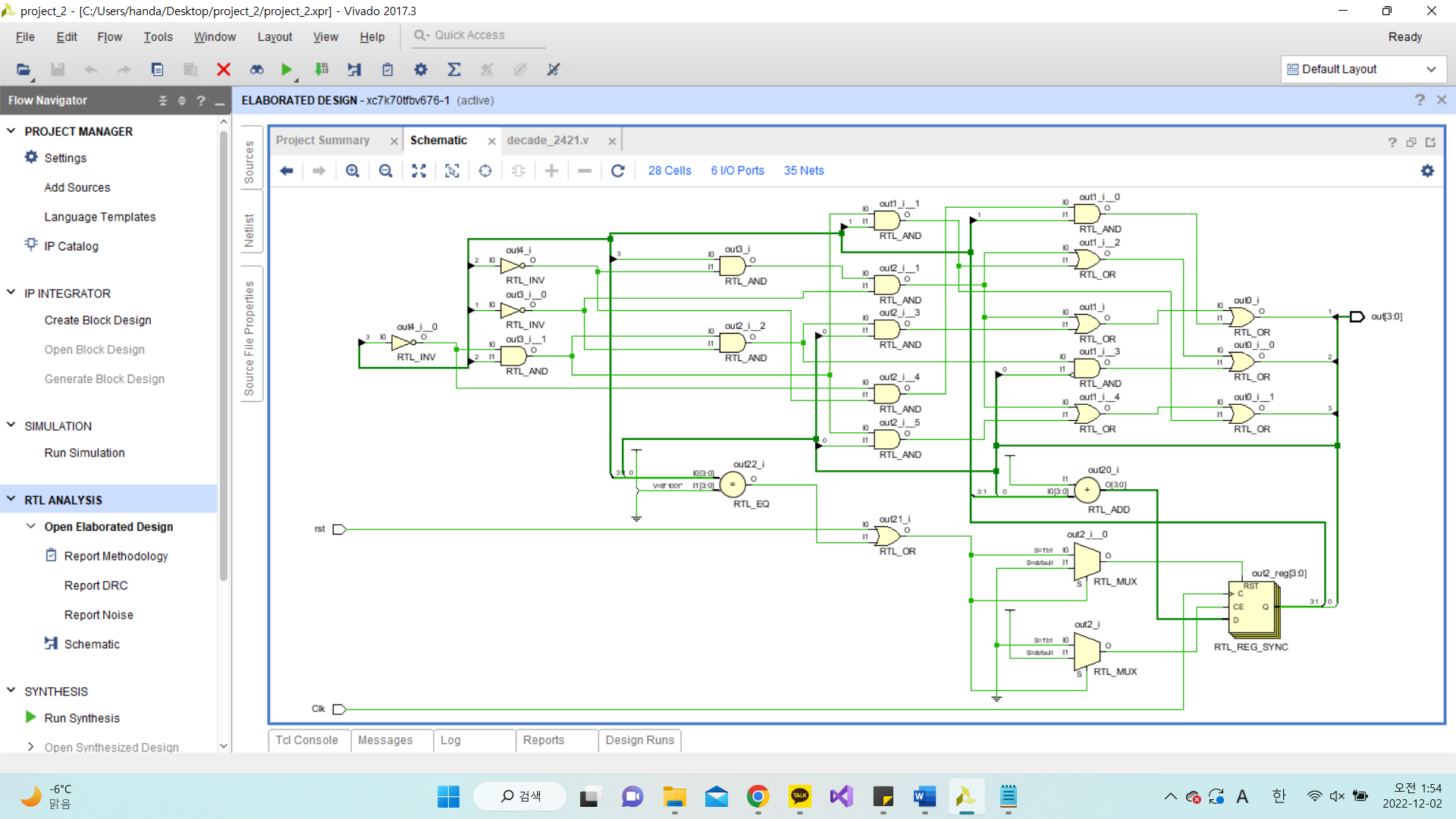
**assign out[3] = (( out2[3] & ~out2[2] & ~out2[1]) | (~out2[3] & out2[2] & out2[0]) | (~out2[3] & out2[2] & out2[1]));**

**assign out[2] = ((out2[3] & ~out2[2] & ~out2[1]) | (~out2[3] & out2[2] & out2[1]) | (~out2[3] & out2[2] & ~out2[1] & ~out2[0]));**

**assign out[1] = ((out2[3] & ~out2[2] & ~out2[1]) | (~out2[3] & out2[2] & ~out2[1] & out2[0]) | (~out2[3] & ~out2[2] & out2[1]));**

**assign out[0] = out2[0];**

**이 코드를 통해 0~9를 카운트한 8421 코드를 2421 코드로 변환할 수 있다. 이를 schematic으로 확인하면 다음과 같다.**

****

**이제 입력의 변화에 따라 출력이 변화하는 것을 확인하기 위해 시뮬레이션 코드를 작성해야 하는데, 전체적인 코드는 다음과 같다.**

**module decade\_2421\_tb;**

**reg Clk, rst;**

**wire[3:0]out;**

**decade\_2421 decade\_2421\_tb(**

**.Clk(Clk),**

**.rst(rst),**

**.out(out));**

**initial begin**

**Clk = 1'b0;**

**rst = 1'b0;**

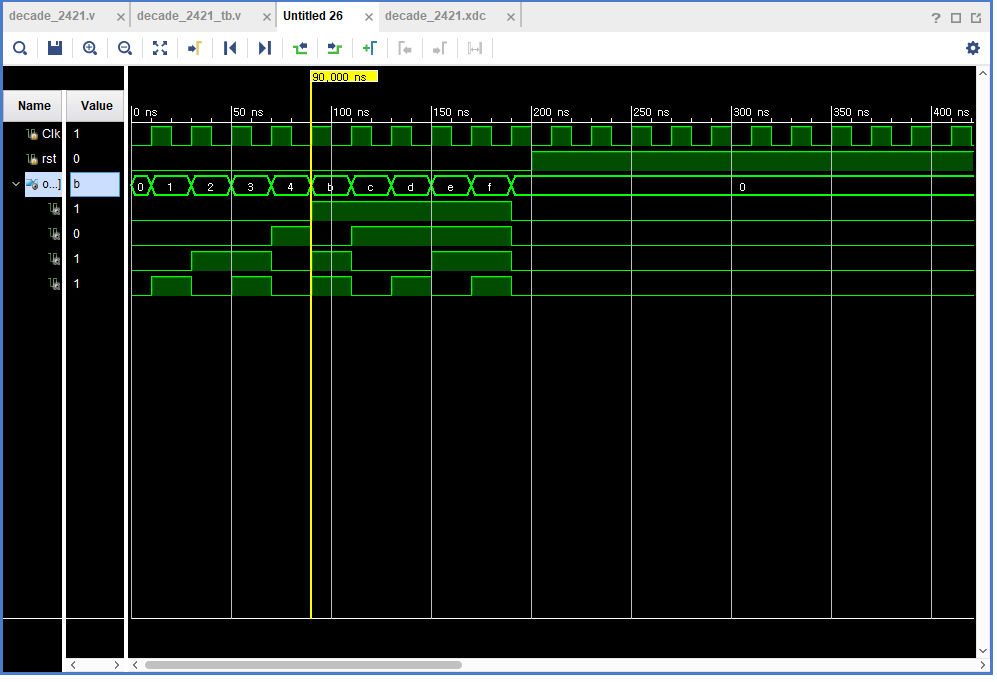
**end**

**always #10 Clk = ~Clk;**

**initial rst <= #200 1'b1;**

**endmodule**

**시뮬레이션 코드는 4 bit decade counter와 동일하다. 입력 Clk, rst와 출력 out을 시뮬레이션 코드에 연결시킨 뒤 Clk의 초기 값을 0으로, rst의 초기 값을 0으로 설정한다. 이 후에 Clk는 일정 간격에 따라 반전되도록 설계하고, rst는 일정 시간이 지난 뒤에 1로 변화하도록 설정한다. 이 코드를 실행하여 시뮬레이션 결과를 확인하면 다음과 같다.**

****

**위 시뮬레이션 결과에서 보여지는 출력 예시는 Clk가 0에서 1로 상승하는 시점이고, rst의 값이 0이며 카운터의 이전 상태는 0100이다. Clk가 0에서 1로 상승하는 상승 에지이기 때문에 카운터가 동작하며, rst의 값이 0이기 때문에 카운터의 상태가 초기화되지 않고, out에 1이 더해진다. 8421 코드에서 0100에 1을 더하면 십진수 5를 나타내는 0101이 되는데, 8421 코드 0101을 2421로 변환하면 1011이 된다. 2421 코드 1011은 십진수로 5를 의미하는데, 이는 8421 코드로 b를 의미하기 때문에 시뮬레이션 결과가 올바르게 출력된 것을 볼 수 있다. 다시 상승에지가 입력된 시점에서 rst의 값은 여전히 0이기 때문에 out2에 1이 더해지는데, 이전 out2의 상태는 0101이기 때문에 out2는 0110이 된다. 이를 2421 코드로 변환하면 1100이 되고, 이는 십진수로 6을, 8421 코드 0110을 의미하기 때문에 역시 올바른 값을 출력하고 있다. 이 후에 200ns가 지나는 시점에서 rst의 값이 1로 변화하는데, rst가 1이 되면 counter가 초기 상태로 돌아가기 때문에 out은 0000이 된다.**

**4.**

**이번 실험에서는 2 bit counter, 4 bit decade counter, 그리고 4 bit 2421 decade counter를 설계하였다. 2 bit counter는 0부터 3까지의 수를 셀 수 있고, 4 bit decade counter와 4 bit 2421 decade counter는 0부터 9까지의 수를 셀 수 있었다. 세 종류의 카운터 모두 reset 입력이 존재하였는데, reset이 1의 값을 가질 때, 카운터의 상태가 초기 상태로 돌아온다. 또한 이번 실험에서는 positive edge일 때 카운터가 트리거 되도록 설정하였다. Positive edge는 clock pulse가 0에서 1로 상승하는 시점에서 카운터가 동작하는 방식이다. Positive edge에서 reset의 값이 1이라면 카운터의 상태가 초기화되고, reset의 값이 0이라면 카운터의 상태가 1이 증가한다. 4 bit decade counter에서 4비트로 구성할 수 있는 상태는 0000에서 1111까지 총 16가지이지만 decade counter는 0부터 9까지의 수, 즉 0000부터 1001까지의 상태만 필요하다. 따라서 1001에서 더 이상 1이 더해지면 안되기 때문에 1001 상태가 되면 다음 상태는 초기 상태인 0000으로 돌아간다. 4 bit 2421 decade counter도 마찬가지로 0부터 9까지의 수를 센다. 하지만 출력 out이 2421 코드를 사용하기 때문에 8421 코드를 2421 코드로 변환하는 과정이 필요하다. 이를 카르노 맵을 사용해 out의 각각의 비트를 설계하였다. 추가적으로, 2421 코드 외에도 5421 코드가 있는데, 8421 코드를 2421로 변환하여 상태를 결정하는 카운터를 설계할 수 있듯이, 비슷한 방식으로 8421을 5421 코드로 변환하여 상태를 결정하는 카운터 역시 설계할 수 있을 것이라고 생각한다.**

**5.**

**실험에서 설계한 decade counter를 사용해 디지털 시계를 설계하는 것이 가능하다. 디지털 시계는 시, 분, 초 세 부분으로 나뉘어져 있으며 분과 초는 0부터 59까지, 시는 0부터 23까지 카운트한다. 따라서 분, 초를 세는 카운터는 MOD-60 카운터를 사용하고, 시를 세는 카운터는 MOD-24 카운터를 사용한다. MOD-60 카운터의 경우 decade counter와 MOD-6 카운터로 이루어져 있는데, 십진 카운터로 1의 자리 0부터 9까지 카운트하고, MOD-6 카운터로 0부터 5까지 카운트한다. 이제 각각의 카운터에 clock pulse를 입력해야 하는데, 초를 나타내는 MOD-60 카운터에는 초당 1 펄스를 가해주고, 분을 나타내는 카운터에는 분당 1 펄스를 가해주며, 시를 나타내는 카운터에는 시간당 1 펄스를 가해주면 디지털 시계를 작동시킬 수 있다.**