12주차 결과보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20212022 이름: 이예준

**1.**

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*2-bit Counter Simulation 결과

위의 사진은 2-bit Counter를 시뮬레이션 돌린 결과이다. 사진을 보게 되면 x가 1이

될 때마다 결과가 바뀌는 것을 보아 x가 Flipflop의 Clock Pulse역할을 하고, rising edge

trigger임을 알 수 있다. 또한 rst가 0일 때는 결과가 계속 0이다가 rst가 1일 때만

Counter의 기능을 하는 것을 보아 rst가 Output을 초기화하는 역할을 하는 것을

알 수 있고 Active Low인 것도 알 수 있다. Q가 0~3까지 올라갔다가 다시 0부터 올라가며

Counter 결과가 정상적으로 나오는 것을 확인할 수 있고, 이런 각 경우에 대한 진리표는

아래와 같다.

|  |  |  |
| --- | --- | --- |
| now next | x=0 | x=1 |
| 00 | 00 | 01 |
| 01 | 01 | 10 |
| 10 | 10 | 11 |
| 11 | 11 | 00 |

\*2-bit Counter 진리표

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*2-bit Counter Verilog 코드

이 Verilog 코드를 통해 위에 봤던 Simulation 결과를 얻을 수 있었다. 이 코드는

2-bit Counter이기 때문에 output Q는 [1:0]배열로 표현했으며, x와 rst는 input으로

선언해주었다. poseedge를 이용해 x가 1이 될 때마다 begin~end 문을 수행하도록 했다.

만약 rst가 0이면 결과값 Q는 0으로 초기화되며, 그 이외의 경우에는 결과값 Q에 1씩

더한다. 코드를 보면 Q가 112일 때 다시 002으로 초기화하는 조건문이 없는데 그 이유는

112 다음이 1002이기 때문에 어차피 결과값 Q배열로는 2-bit만 표현 할 수 있어 002가

된다. 따라서 굳이 002으로 초기화하는 조건은 필요 없다. simulation 코드에서는 x를

10클럭마다 반전시켜주었고, rst는 200클럭마다 반전시켜주어 rst가 0일 때의 Q의 모든

경우의 수와 rst가 1일 때의 Q의 모든 경우의 수를 볼 수 있게 해주었다. 위에 있는

Simulation 결과를 보면 알 수 있다.

도표, 평면도, 텍스트, 기술 도면이(가) 표시된 사진

자동 생성된 설명

\*2-bit Counter Schematic 결과

이 사진은 Verilog 코드에 대한 Schematic이다.

x는 RTL\_REG의 C(clock)과 연결하여 x가 trigger될 때마다 RTL\_REG가 작동하도록 되어 있고, rst는 각 Mutiplexer의 Select 선에 연결하여 RTL\_REG를 초기화할지 결정한다.

출력값 Q는 RTL\_ADD를 통해 1씩 더한다.

**2.**

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*4-bit Decade Counter Simulation 결과

위의 사진은 4-bit Decade Counter를 시뮬레이션 돌린 결과이다. 위에서 설명한

2-bit Counter와 비슷하게 1씩 올라가는 Counter이지만 차이점은 11112까지 가지 않고

0~9까지만 Counting하는 것이다. 따라서 10012이 되면 다시 00002부터 올라간다.

사진을 보게 되면 2-bit Counter와 똑같이 x가 rising edge trigger이며, rst도 Output을

초기화하는 역할과 Active Low인 것도 알 수 있다. Q가 0~9까지 올라갔다가 다시 0부터

올라가며 Counter 결과가 정상적으로 나오는 것을 확인할 수 있고, 이런 각 경우에 대한 진리표는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| now next | x=0 | x=1 |
| 0000 | 0000 | 0001 |
| 0001 | 0001 | 0010 |
| 0010 | 0010 | 0011 |
| 0011 | 0011 | 0100 |
| 0100 | 0100 | 0101 |
| 0101 | 0101 | 0110 |
| 0110 | 0110 | 0111 |
| 0111 | 0111 | 1000 |
| 1000 | 1000 | 1001 |
| 1001 | 1001 | 0000 |

\*4-bit Decade Counter 진리표

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*4-bit Decade Counter Verilog 코드

이 Verilog 코드를 통해 위에 봤던 Simulation 결과를 얻을 수 있었다. 이 코드는

4-bit Decade Counter이기 때문에 output Q는 [3:0]배열로 표현했으며, x와 rst는 input으로

선언해주었다. poseedge를 이용해 x가 1이 될 때마다 begin~end 문을 수행하도록 했다.

만약 rst가 0이면 결과값 Q는 0으로 초기화된다. 2-bit Counter는 112를 넘어가면 Carry가

발생해 자동으로 002으로 초기화됐지만 이 Counter의 경우 0부터 9까지 올라간 뒤

더 올라가지 않고 00002로 초기화시켜야 되기 때문에 Q가 10012이 될 때 0으로

초기화하는 조건문이 필요하다. simulation 코드에서는 x를 10클럭마다 반전시켜주었고,

rst는 400클럭마다 반전시켜주어 rst가 0일 때의 Q의 모든 경우의 수와 rst가 1일 때의

Q의 모든 경우의 수를 볼 수 있게 해주었다. 위에 있는 Simulation 결과를 보면 알 수 있다.

도표, 라인, 평면도, 텍스트이(가) 표시된 사진

자동 생성된 설명

\*4-bit Decade Counter Schematic 결과

이 사진은 Verilog 코드에 대한 Schematic이다.

x는 RTL\_REG의 C(clock)과 연결하여 x가 trigger될 때마다 RTL\_REG가 작동하도록 되어 있고, rst는 각 Multiplexer의 Select 선에 연결하여 RTL\_REG를 초기화할지 결정한다.

출력값 Q는 RTL\_ADD를 통해 1씩 더한다.

**3.**

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*4-bit 2421 Decade Counter Simulation 결과

위의 사진은 4-bit 2421 Decade Counter를 시뮬레이션 돌린 결과이다. 위에서 설명한

4-bit Decade Counter와 비슷하게 0~9까지 올라가는 Counter이지만 기존 8421코드가

아니라 2421코드라는 차이점이 있다. 이름을 통해서 간단하게 어떤 차이점이 있는지

이해할 수 있는데 각 bit가 의미하는 수가 다른 것이다. 예를 들어 11012를 8421코드로

보면 8:1, 4:1, 2:0, 1:1이 되어 십진수로 13이 되지만 2421코드를 보면 2:1, 4:1, 2:0, 1:1이

되어 십진수로 7이 된다. 사진을 보게 되면 위에 있는 다른 Counter와 똑같이 x가

rising edge trigger이며, rst도 Output을 초기화하는 역할과 Active Low인 것도 알 수 있다.

하지만 Q를 보게 되면 1부터 4까지 잘 올라가다가 중간을 건너뛰고 b(11)이 되는데

그 이유는 2421코드로 구현했으나 simulation은 8421 코드로 해석해서 표현했기 때문이다. 0~9까지 올라갔다가 다시 0부터 올라가며 Counter 결과가 정상적으로 나오는 것을

확인할 수 있고, 이런 각 경우에 대한 진리표는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| now next | x=0 | x=1 |
| 0000 | 0000 | 0001 |
| 0001 | 0001 | 0010 |
| 0010 | 0010 | 0011 |
| 0011 | 0011 | 0100 |
| 0100 | 0100 | 1011 |
| 1011 | 1011 | 1100 |
| 1100 | 1100 | 1101 |
| 1101 | 1101 | 1110 |
| 1110 | 1110 | 1111 |
| 1111 | 1111 | 0000 |

\*4-bit 2421 Decade Counter 진리표

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

\*4-bit 2421 Decade Counter Verilog 코드

이 Verilog 코드를 통해 위에 봤던 Simulation 결과를 얻을 수 있었다. 이 코드는

4-bit 2421 Decade Counter 이기 때문에 output Q는 [3:0]배열로 표현했으며, x와 rst는

input으로 선언해주었다. poseedge를 이용해 x가 1이 될 때마다 begin~end 문을

수행하도록 했다. 만약 rst가 0이면 결과값 Q는 0으로 초기화된다. 위의 진리표를 보게

되면 2421코드 기준으로는 11112이 9가 되기 때문에 4-bit Decade Counter에서 10012이

되면 0으로 초기화하는 과정이 필요하지 않다. 단, Q를 1씩 올리는 과정이 8421 기준으로

올리기 때문에 2421코드로 맞춰 주기 위해 Q가 01002일 때 7을 더해 11012로 바꿔준다.

왜냐하면 2421 코드 기준으로 0100이 4이고, 1101이 5이기 때문이다.

simulation 코드에서는 x를 10클럭마다 반전시켜주었고, rst는 400클럭마다 반전시켜주어

rst가 0일 때의 Q의 모든 경우의 수와 rst가 1일 때의 Q의 모든 경우의 수를 볼 수 있게

해주었다. 위에 있는 Simulation 결과를 보면 알 수 있다.

**4.**

이번 실험을 통해 Counter에 대해 Verliog를 이용한 구현 방법에 대해 알아보고

Simulation결과와 FPGA 보드 구현 결과가 정상적으로 나오는 것을 확인할 수 있었다.

또한 지금까지 실험에서의 입력값을 FPGA보드의 toggle switch와 연결했지만 이번

실험에서는 trigger 역할을 하는 x를 FPGA보드의 push switch와 연결했다는 차이점이 있다.

2-bit Counter는 별다른 조건없이 Counting하는 기능과 초기화하는 기능으로 구현했고,

4-bit Decade Counter는 0~9까지 Counting하기 위해 10012에서 초기화하는 기능을 추가했다. 4-bit 2421 Decade Counter 역시 0~9까지 Counting하지만 8421코드를 2421코드로 변경하기 위해 01002에서 7을 더해 11012로 바꿔주는 기능을 추가했다.

이번 실험에서의 논의사항으로는 Decade나 2421코드 외의 다른 형식의 코드

구현 가능성이다. 2421코드외에도 3초과 코드나 84-2-1코드 등도 조건문을 통해 간단히

구현할 수 있겠다는 생각이 든다.

-----------------------------

-3초과 코드 조건문

always@(poseedge x)

begin

if(!rst)

Q <= 0;

else if(Q ==4’b1100)

Q<= 0;

else

-----------------------------

**5.**

Counter는 특정 사건이나 과정이 발생한 횟수를 저장하는 장치이다. 따라서 주로 Clock

Signal에 따라 Counting에 사용된다. 이때 초기 상태에 상관없이 Counter loop에

들어갈 수 있으면 스스로 시작한다고 할 수 있는데, 어떤 상태에서도 가장 긴 loop를

돌 수 있는 카운터를 Self-Starting Counter라고 한다. Self-Starting Counter는 trap 상태를 피할 수 있도록 만들어졌는데, trap상태란 Counter 오류로 잘못된 곳으로 접근한 상태를 말하며, Self-Starting Counter가 자체적으로 Main Counting loop로 돌아올 수 있다.

텍스트, 도표, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

\*각종 Counter loop 경로 그림

만약 왼쪽 그림에서 4부터 출발한다면, 4≫0≫1≫2≫3≫4 이렇게 Main loop를 돌지만,

5, 6 또는 7에서 출발한다면 Main loop로 돌아올 수 없다. 하지만 오른쪽 그림에서 4부터

출발한다면, 4≫0≫1≫2≫3≫4 이렇게 Main loop를 돌며 5, 6, 또는 7에서 출발한다 해도

0과 연결되어 있기 때문에 언제든 Main loop로 돌아올 수 있다.