12주차 결과보고서

전공: 컴퓨터공학과 학년: 4학년 학번: 20191583 이름: 김태곤

**1. 2-bit counter 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(verilog source code, 출력 예시/schematic 포함, 과정 상세히 적을것)**

**1) State Diagram**

**도표, 텍스트, 원, 라인이(가) 표시된 사진

자동 생성된 설명**

**2) Truth Table**

|  |  |  |
| --- | --- | --- |
| Present State Q | Next State Q\* | |
| input x = 0 | input x = 1 |
| 00 | 00 | 01 |
| 01 | 01 | 10 |
| 10 | 10 | 11 |
| 11 | 11 | 00 |

**3) Source Code**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4) Schematic**

**도표, 스크린샷, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명**

**5) Simulation**

**소프트웨어, 멀티미디어 소프트웨어, 텍스트, 라인이(가) 표시된 사진

자동 생성된 설명**

**6) 결과 및 과정**

BinaryCounter 모듈은 2bit binary counter를 구현한다. 이 모듈의 입력 포트로는 리셋 신호인 reset과 클럭 신호인 clk가 있고, 출력 포트로는 2bit의 현재 카운터 값을 나타내는 out이 있다. 초기값 설정을 위해 initial out = 2'b00;를 통해 카운터의 초기값을 0으로 설정한다. 항상 블록인 always @(posedge clk or posedge reset)은 클럭 신호의 상승 에지나 리셋 신호의 상승 에지에서 동작한다. 리셋 신호가 활성화되면 out <= 2'b00;을 통해 카운터를 0으로 초기화하고, 리셋 신호가 비활성화된 상태에서 클럭 신호가 상승 에지에 도달하면 out <= out + 2'b01;을 통해 카운터 값을 1 증가시킨다.

BinaryCounter 모듈은 2bit binary counter를 구현한다. 이 모듈의 입력 포트로는 리셋 신호인 reset과 클럭 신호인 clk가 있고, 출력 포트로는 2bit의 현재 카운터 값을 나타내는 out이 있다. 초기값 설정을 위해 initial out = 2'b00;를 통해 카운터의 초기값을 0으로 설정한다. 항상 블록인 always @(posedge clk or posedge reset)은 클럭 신호의 상승 에지나 리셋 신호의 상승 에지에서 동작한다. 리셋 신호가 활성화되면 out <= 2'b00;을 통해 카운터를 0으로 초기화하고, 리셋 신호가 비활성화된 상태에서 클럭 신호가 상승 에지에 도달하면 out <= out + 2'b01;을 통해 카운터 값을 1 증가시킨다.

시뮬레이션은 시작 후 150ns 동안 카운터는 리셋 신호가 0이므로 잘 동작한다. 150ns 시점에 리셋 신호가 1로 설정되어 카운터는 0으로 초기화된다. 200ns 시점에 시뮬레이션이 종료된다. 이 코드는 2bit binary counter가 정상적으로 동작하는지 확인하기 위해 설계되었으며, 리셋 신호가 들어오면 카운터를 0으로 초기화하고, 리셋 신호가 0일 때는 클럭 신호의 상승 에지에서 카운터를 1씩 증가시키는 것을 확인할 수 있다.

시뮬레이션 결과를 살펴보면 State Table에 맞는 결과가 나온것을 확인할 수 있다. 따라서 2bit binary counter에 대한 구현을 잘 이루어졌다고 볼 수 있다.

**2. 4-bit decade counter의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(verilog source code, 출력 예시/schematic 포함, 과정 상세히 적을것)**

**1) State Diagram**

**도표, 텍스트, 원, 폰트이(가) 표시된 사진

자동 생성된 설명**

**2) Truth Table**

|  |  |  |
| --- | --- | --- |
| Present State Q | Next State Q\* | |
| input x=0 | input 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 |

**3) Source Code**

**텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명**

**4) Schematic**

**도표, 텍스트, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명**

**5) Simulation**

**소프트웨어, 텍스트, 멀티미디어 소프트웨어, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**6) 결과 및 과정**

DecadeCounter 모듈은 4bit decade counter를 구현한다. 이 모듈의 입력 포트로는 리셋 신호인 reset과 클럭 신호인 clk가 있고, 출력 포트로는 4bit의 현재 카운터 값을 나타내는 out이 있다. 초기값 설정을 위해 initial out = 4'b0000;를 통해 카운터의 초기값을 0으로 설정한다. 항상 블록인 always @(posedge clk or posedge reset)은 클럭 신호의 상승 에지나 리셋 신호의 상승 에지에서 동작한다. 리셋 신호가 활성화되면 out <= 4'b0000;을 통해 카운터를 0으로 초기화하고, 리셋 신호가 비활성화된 상태에서 클럭 신호가 상승 에지에 도달하면, 카운터가 9(즉, 4'b1001)에 도달했을 때 out <= 4'b0000;으로 리셋하고, 그렇지 않으면 out <= out + 4'b0001;을 통해 카운터 값을 1 증가시킨다.

DecadeCounter\_tb 모듈은 DecadeCounter 모듈을 테스트하기 위한 테스트 벤치이다. 이 모듈의 입력 포트로는 클럭 신호를 생성하기 위한 레지스터 clk와 리셋 신호를 생성하기 위한 레지스터 reset이 있으며, 출력 포트로는 DecadeCounter 모듈의 출력을 받기 위한 와이어 out이 있다. DecadeCounter 모듈을 인스턴스화하여 DecadeCounter u\_DecadeCounter를 통해 해당 모듈의 포트와 테스트 벤치의 신호를 연결한다. 초기화 블록에서는 clk = 1'b0;을 통해 클럭 신호를 0으로, reset = 1'b0;을 통해 리셋 신호를 0으로 초기화하고, 시뮬레이션 시작 450ns 후에 리셋 신호를 1로 설정한다. 클럭 생성 블록은 always begin clk = #10 ~clk; end를 통해 10ns 주기로 클럭 신호를 반전시켜 클럭 주기를 20ns로 설정한다. 시뮬레이션 종료 블록은 initial begin #500 $finish; end를 통해 500ns 후에 시뮬레이션을 종료한다.

시뮬레이션은 시작 후 450ns 동안 카운터는 리셋 신호가 0이므로 잘 동작한다. 카운터는 0부터 9까지 세고, 10이 되면 다시 0으로 리셋된다. 450ns 시점에 리셋 신호가 1로 설정되어 카운터는 0으로 초기화된다. 500ns 시점에 시뮬레이션이 종료된다. 이 코드는 4bit decade counter가 정상적으로 동작하는지 확인하기 위해 설계되었으며, 리셋 신호가 들어오면 카운터를 0으로 초기화하고, 리셋 신호가 0일 때는 클럭 신호의 상승 에지에서 카운터를 1씩 증가시키는 것을 확인할 수 있다. 시뮬레이션 결과를 살펴보면 State Table에 맞는 결과가 나온 것을 확인할 수 있다. 따라서 4bit decade counter에 대한 구현이 잘 이루어졌다고 볼 수 있다.

**3. 4-bit 2421 decade counter 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(verilog source code, 출력 예시/schematic 포함, 과정 상세히 적을것)**

**1) State Diagram**

**도표, 텍스트, 원, 폰트이(가) 표시된 사진

자동 생성된 설명**

**2) Truth Table**

|  |  |  |
| --- | --- | --- |
| Present State Q | Next State Q\* | |
| input x=0 | input 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 |

**3) Source Code**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**4) Schematic**

**도표, 라인, 평면도, 지도이(가) 표시된 사진

자동 생성된 설명**

**5) Simulation**

**텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**6) 결과 및 과정**

Counter2421 모듈은 4bit 2421 decade counter를 구현한다. 이 모듈의 입력 포트로는 리셋 신호인 reset과 클럭 신호인 clk가 있고, 출력 포트로는 4bit의 현재 카운터 값을 나타내는 out이 있다. 초기값 설정을 위해 initial out = 4'b0000;을 통해 카운터의 초기값을 0으로 설정한다. 항상 블록인 always @(posedge clk or posedge reset)은 클럭 신호의 상승 에지나 리셋 신호의 상승 에지에서 동작한다. 리셋 신호가 활성화되면 out <= 4'b0000;을 통해 카운터를 0으로 초기화하고, 리셋 신호가 비활성화된 상태에서 클럭 신호가 상승 에지에 도달하면, 카운터가 9(즉, 4'b1001)에 도달했을 때 tmp <= 4'b0000;으로 리셋하고, 그렇지 않으면 tmp <= tmp + 4'b0001;을 통해 카운터 값을 1 증가시킨다. 이후 다음 상태를 계산하여 out에 값을 할당한다. out[3] <= tmp[3] | (tmp[2] & tmp[1]) | (tmp[2] & tmp[0]);, out[2] <= tmp[3] | (tmp[2] & tmp[1]) | (tmp[2] & (~tmp[0]));, out[1] <= tmp[3] | ((~tmp[2]) & tmp[1]) | (tmp[2] & (~tmp[1]) & tmp[0]);, out[0] <= tmp[0];.

Counter2421\_tb 모듈은 Counter2421 모듈을 테스트하기 위한 테스트 벤치이다. 이 모듈의 입력 포트로는 클럭 신호를 생성하기 위한 레지스터 clk와 리셋 신호를 생성하기 위한 레지스터 reset이 있으며, 출력 포트로는 Counter2421 모듈의 출력을 받기 위한 와이어 out이 있다. Counter2421 모듈을 인스턴스화하여 Counter2421 u\_Counter2421를 통해 해당 모듈의 포트와 테스트 벤치의 신호를 연결한다. 초기화 블록에서는 clk = 1'b0;을 통해 클럭 신호를 0으로, reset = 1'b0;을 통해 리셋 신호를 0으로 초기화하고, 시뮬레이션 시작 450ns 후에 리셋 신호를 1로 설정한다. 클럭 생성 블록은 always begin clk = #10 ~clk; end를 통해 10ns 주기로 클럭 신호를 반전시켜 클럭 주기를 20ns로 설정한다. 시뮬레이션 종료 블록은 initial begin #500 $finish; end를 통해 500ns 후에 시뮬레이션을 종료한다. 시뮬레이션은 시작 후 450ns 동안 카운터는 리셋 신호가 0이므로 잘 동작한다. 카운터는 0부터 9까지 세고, 10이 되면 다시 0으로 리셋된다. 450ns 시점에 리셋 신호가 1로 설정되어 카운터는 0으로 초기화된다. 500ns 시점에 시뮬레이션이 종료된다. 이 코드는 4bit 2421 decade counter가 정상적으로 동작하는지 확인하기 위해 설계되었으며, 리셋 신호가 들어오면 카운터를 0으로 초기화하고, 리셋 신호가 0일 때는 클럭 신호의 상승 에지에서 카운터를 1씩 증가시키는 것을 확인할 수 있다. 시뮬레이션 결과를 살펴보면 State Table에 맞는 결과가 나온 것을 확인할 수 있다. 따라서 4bit 2421 decade counter에 대한 구현이 잘 이루어졌다고 볼 수 있다.

**4. 결과 검토 및 논의 사항.**

BinaryCounter, DecadeCounter, Counter2421 모듈의 시뮬레이션 결과를 검토한 결과, 모든 모듈이 설계대로 정상적으로 동작함을 확인할 수 있었다. BinaryCounter 모듈은 리셋 신호가 1로 설정되면 카운터가 0으로 초기화된다, 리셋 신호가 0일 때 클럭 신호의 상승 에지에서 카운터 값을 1씩 증가시키는 동작을 수행한다. 시뮬레이션 동안 클럭 신호에 따라 카운터 값이 정상적으로 증가하였으며, 150ns 후 리셋 신호가 1로 설정되어 카운터가 정상적으로 초기화되었다.

DecadeCounter 모듈은 4비트 카운터로, 리셋 신호가 활성화되면 카운터를 0으로 초기화하였다, 리셋 신호가 0일 때 클럭 신호의 상승 에지에서 카운터 값이 1씩 증가하며 9에 도달하면 다시 0으로 리셋된다. 시뮬레이션에서는 450ns 동안 카운터가 정상적으로 동작하였으며, 클럭 신호에 따라 0부터 9까지 정상적으로 카운트되었다. 450ns 시점에 리셋 신호가 1로 설정되어 카운터가 0으로 초기화되었다.

Counter2421 모듈은 4비트 2421 디케이드 카운터로, 리셋 신호가 활성화되면 카운터를 0으로 초기화하였다, 리셋 신호가 0일 때 클럭 신호의 상승 에지에서 카운터 값을 1씩 증가시킨다. 카운터가 9에 도달하면 다시 0으로 리셋되며, 이후 상태를 계산하여 출력을 설정한다. 시뮬레이션 결과, 450ns 동안 카운터는 정상적으로 동작하였고, 클럭 신호에 따라 0부터 9까지 정상적으로 카운트되었다. 450ns 시점에 리셋 신호가 1로 설정되어 카운터가 초기화되었다.

시뮬레이션 결과, 모든 모듈의 동작이 State Table과 일치하는 것을 확인할 수 있었다. 이는 각 모듈이 설계된 대로 정확하게 동작함을 의미한다. BinaryCounter, DecadeCounter, Counter2421 모듈 모두 정상적으로 초기화되고, 클럭 신호에 동기화되어 카운트가 증가하거나 리셋되는 동작을 확인할 수 있었다. 따라서, 이들 모듈은 디지털 시스템에서 신뢰성 있게 동작할 것으로 판단된다. 추가적으로 다양한 테스트 조건에서도 안정적인 동작을 확인하는 것이 필요할 것이다. 예를 들어, 연속적인 리셋 신호 입력, 빠른 클럭 신호 변화 등의 다양한 시나리오에서 모듈의 동작을 확인하여 더욱 신뢰성 있는 설계를 보장할 수 있다. 이러한 추가 검증을 통해 모듈의 성능을 최적화하고 다양한 응용 환경에서 안정적인 동작을 기대할 수 있을 것이다.

**5. 추가 이론 조사 및 작성.**

피보나치 카운터(Fibonacci Counter)는 피보나치 수열을 따르는 특수 목적의 카운터이다. 피보나치 수열은 첫 두 항이 0과 1이며, 그 이후의 모든 항이 바로 앞의 두 항을 더한 값으로 정의된다. 수학적으로 표현하면 다음과 같다. F(n) = F(n-1) + F(n-2), F(1)=1이다. 피보나치 카운터는 이러한 피보나치 수열의 원리를 디지털 회로로 구현한 것이다.

피보나치 카운터의 구조는 일반적인 이진 카운터와는 다르며, 피보나치 수열을 생성하기 위해 더 복잡한 회로가 필요하다. 기본적인 구성 요소로는 플립플롭(flip-flop), 가산기(adder), 멀티플렉서(multiplexer) 등이 사용된다. 피보나치 수열을 계산하기 위해 두 개의 레지스터를 사용하여 이전 두 값을 저장하고, 이를 더하여 다음 값을 생성한다.

피보나치 카운터의 회로 설계는 여러 주요 블록으로 구성된다. 첫 번째로, 각 레지스터는 플립플롭(flip-flop)으로 구성되며, 이는 현재 피보나치 수열의 값을 저장하는 역할을 한다. 피보나치 수열을 생성하려면 두 개의 값을 유지하고 업데이트할 필요가 있는데, 이 두 값을 플립플롭을 사용하여 저장한다.

두 번째로, 가산기(adder)가 필요하다. 가산기는 두 개의 레지스터 값을 더하여 새로운 피보나치 수를 생성한다. 예를 들어, 이전 두 값이 각각 1과 2라면 가산기는 이 두 값을 더하여 3을 생성한다. 이 결과는 다음 상태로 전이될 때 사용된다.

세 번째로, 멀티플렉서(multiplexer)가 포함된다. 멀티플렉서는 새로운 피보나치 값을 선택하고 이를 다음 클럭 사이클에서 레지스터로 전송하는 역할을 한다. 즉, 가산기에서 생성된 새로운 피보나치 값을 선택하여 레지스터로 이동시키는 경로를 설정해준다. 이를 통해 다음 클럭 주기에서 사용할 값이 적절히 선택되고 저장될 수 있다.

마지막으로, 제어 논리(control logic)가 필요하다. 제어 논리는 클럭 신호와 리셋 신호를 관리하여 카운터의 상태를 제어한다. 예를 들어, 리셋 신호가 활성화되면 카운터는 초기 상태로 돌아가야 한다. 또한, 클럭 신호에 따라 레지스터의 값이 업데이트되어야 한다. 제어 논리는 이러한 신호들을 조정하여 피보나치 카운터가 정확하게 동작하도록 한다.

스케치, 그림, 원, 도표이(가) 표시된 사진

자동 생성된 설명