13주차 결과보고서

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

**1. 4-bit Shift Register의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(verilog source, 출력 예시, 과정 상세히 적을것!)**

**1) Source Code**

**텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 디스플레이이(가) 표시된 사진

자동 생성된 설명**

**2) Output Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Shift Register OUTPUT TABLE** | | | | | |
| **Number Clock Transitions** | **OUTPUTS** | | | | |
| **↑** | **IN** | **L1** | **L2** | **L3** | **L4** |
| **1** | **Reset** | **0** | **0** | **0** | **0** |
| **2** | **1** | **1** | **0** | **0** | **0** |
| **3** | **0** | **0** | **1** | **0** | **0** |
| **4** | **1** | **1** | **0** | **1** | **0** |
| **5** | **0** | **0** | **1** | **0** | **1** |
| **6** | **1** | **1** | **0** | **1** | **0** |
| **7** | **1** | **1** | **1** | **0** | **1** |

**3) Schematic**

**도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

**4) Simulation**

**스크린샷, 텍스트, 사각형이(가) 표시된 사진

자동 생성된 설명**

**5) 실험 과정 및 결과**

Shift Register 모듈은 clk, reset, in을 input값으로 가지고, L[3]를 output값으로 가진다. clk 신호가 1인 경우, 입력된 다른 정보에 의해 output값이 변화한다. 이때 reset이 1인 경우, output L[3], L[2], L[1], L[0]는 0으로 초기화되고, reset이 0인 경우 output값은 clk신호에 따라 shift한다. L[3]의 값은 input in의 값에따라 1인 경우 1을 추가하여 shift하였고, 0인경우 L[3]의 값을 0으로 지정하여 shift하였다. (표에는 L[3], L[2], L[1], L[0]를 L1, L2, L3, L4로 나타내었다.)

testbench에서는 in의 입력값을 reset,101011순으로 입력하였다. clk의 값은 50마다 변화하며, 초기 reset의 값을 70만큼 1로 지정하여 초기화를 시켰다. 이후 clk가 활성화될 때마다 101011순으로 in의 값이 변화하고, 위 simulation과 표와 같은 결과를 얻을 수 있었다.

simulation 표를 살펴보면 in의 값이 1인 경우 L[3]에 성공적으로 1의 값이 들어왔고, L[2], L[1], L[0]는 한칸 shift한 것을 확인할 수 있다. 또한 0인 경우, L[3]에는 0이 들어오며, 동일하게 shift한 것을 확인할 수 있다.

따라서 실습 결과 성공적으로 코드를 구현하였음을 알 수 있다.

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

**(verilog source, 출력 예시, 과정 상세히 적을것!)**

**1) Source Code**

**텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명**

**2) Output Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ring Counter OUTPUT TABLE** | | | | |
| **Number Clock Transitions** | **OUTPUTS** | | | |
| **↑** | **L1** | **L2** | **L3** | **L4** |
| **1** | **1** | **0** | **0** | **0** |
| **2** | **0** | **1** | **0** | **0** |
| **3** | **0** | **0** | **1** | **0** |
| **4** | **0** | **0** | **0** | **1** |
| **5** | **1** | **0** | **0** | **0** |
| **6** | **0** | **1** | **0** | **0** |
| **7** | **0** | **0** | **1** | **0** |

**3) Schematic**

**도표, 라인, 평면도, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**4) Simulation**

**텍스트, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명**

**5) 실험 과정 및 결과**

Ring Counter 모듈은 clk, reset을 input값으로 가지고, L[3]를 output값으로 가진다. clk 신호가 1인 경우, output값이 변화한다. 이때 reset이 1인 경우, output L[3], L[2], L[1], L[0]는 0으로 초기화되고, reset이 0인 경우 output값은 clk신호에 따라 L의 값이 변화한다. L의 값은 하나만 1이고 나머지는 0이다. clk 신호에 따라 L의 값은 한칸씩 옆으로 이동하게 된다. (표에는 L[3], L[2], L[1], L[0]를 L1, L2, L3, L4로 나타내었다.)

testbench에서는 clk의 값을 50마다 변화시켰다. 초기 L의 값을 1000으로 지정하였고, 이후 clk가 활성화될 때마다 0100, 0010, 0001순으로 L의 값이 변화하게 구성하였다. 또한 reset의 값이 1인 겨우 0000으로 초기화시켰다. 구현 결과 위 simulation과 표와 같은 결과를 얻을 수 있었다.

simulation 표를 살펴보면 clk의 활성에 따라 L[3], L[2], L[1], L[0]는 한칸 shift한 것을 확인할 수 있다. 1000, 0100, 0010, 0001을 반복하여 순환하였다. 또한 reset이 0인 경우, L의 값이 0으로 초기화 된것을 확인할 수 있다.

따라서 실습 결과 성공적으로 코드를 구현하였음을 알 수 있다.

**3. 4-bit Up/Down counter 의 결과 및 Simulation 과정에 대해서 설명하시오.**

**(verilog source, 출력 예시, 과정 상세히 적을것!)**

**1) Source Code**

**텍스트, 스크린샷, 번호, 디스플레이이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명**

**2) Output Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Up Counter OUTPUT TABLE** | | | | | |
| **Number Clock Transitions** | **OUTPUTS** | | | | |
| **↑** | **L1** | **L2** | **L3** | **L4** | **DISPLAY** |
| **1** | **0** | **0** | **0** | **1** | **U** |
| **2** | **0** | **0** | **1** | **0** | **U** |
| **3** | **0** | **0** | **1** | **1** | **U** |
| **4** | **0** | **1** | **0** | **0** | **U** |
| **5** | **0** | **1** | **0** | **1** | **U** |
| **6** | **0** | **1** | **1** | **0** | **U** |
| **7** | **0** | **1** | **1** | **1** | **U** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Down Counter OUTPUT TABLE** | | | | | |
| **Number Clock Transitions** | **OUTPUTS** | | | | |
| **↑** | **L1** | **L2** | **L3** | **L4** | **DISPLAY** |
| **1** | **0** | **1** | **1** | **1** | **D** |
| **2** | **0** | **1** | **1** | **0** | **D** |
| **3** | **0** | **1** | **0** | **1** | **D** |
| **4** | **0** | **1** | **0** | **0** | **D** |
| **5** | **0** | **0** | **1** | **1** | **D** |
| **6** | **0** | **0** | **1** | **0** | **D** |
| **7** | **0** | **0** | **0** | **1** | **D** |

**3) Schematic**

**도표, 라인, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

**4) Simulation**

**텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명**

**5) 실험 과정 및 결과**

UpDownCounter 모듈은 clk, reset, up을 input값으로 가지고, L[3], seg[6]를 output값으로 가진다. clk 신호가 1인 경우, 입력된 다른 정보에 의해 output값이 변화한다. 이때 reset이 1인 경우, output L[3], L[2], L[1], L[0]는 0으로 초기화되고, seg의 값또한 0으로 초기화하였다. reset이 0이고, up의 값이 1인 경우 output값은 clk신호에 따라 1씩 증가하였다. 이때 seg의 값은 U(0111110)를 나타낸다. up의 값이 0인 경우 output값은 clk 신호에 따라 1씩 감소하였다. 이때 seg의 값은 d(1011110)을 나타낸다.

testbench에서는 clk의 값을 50마다 변화시켰다. 초기 reset을 통해 L과 seg를 초기화하였다. 이후 L과 seg의 값은 up의 신호에 따라 증가하거나 감소하게 구성하였다. 구현 결과 위 simulation과 표와 같은 결과를 얻을 수 있었다.

simulation 표를 살펴보면 clk의 활성과 up=1의 값에 따라 L의 값은 1씩 증가하였으며, up=0일때는 1씩 감소하였다. 이에 맞게 seg의 값도 성공적으로 출력되었다.

따라서 실습 결과 성공적으로 코드를 구현하였음을 알 수 있다.

**4. 결과 검토 및 논의 사항.**

Shift Register 모듈의 구현에서는 클럭 신호와 입력 신호에 따른 출력을 확인하였다. 이 모듈은 clk, reset, in을 입력으로 받아 L[3]을 출력으로 제공한다. 클럭 신호가 1인 경우, 입력 신호에 따라 출력 값이 변하는 것을 관찰할 수 있었다. reset 신호가 1일 때는 출력 값 L[3], L[2], L[1], L[0]가 0으로 초기화되며, reset 신호가 0일 때는 clk 신호에 따라 값을 시프트한다. 특히, 입력 in의 값이 1일 때는 L[3]의 값이 1로 설정되고, 입력 in의 값이 0일 때는 L[3]의 값이 0으로 설정된 후 나머지 값들이 한 칸씩 시프트되었다. 이 과정은 시뮬레이션 표에서도 확인할 수 있었으며, 입력 신호에 따라 성공적으로 시프트 레지스터가 동작함을 알 수 있었다. in 값이 "101011" 순으로 입력되었을 때, 시뮬레이션 결과 L[3]에 1 또는 0이 들어오며 적절히 시프트 되는 것이 확인되었고, 결과적으로 코드는 정확하게 구현되었음을 확인할 수 있었다.

Ring Counter 모듈의 구현에서는 클럭 신호와 초기화 상태에서 출력이 순환적으로 변화하는 동작을 확인하였다. 이 모듈은 clk와 reset을 입력으로 받아 L[3]을 출력으로 제공한다. 클럭 신호가 1일 때, 출력 값이 순환적으로 변화하는 것을 확인할 수 있었다. reset 신호가 1일 때는 L[3], L[2], L[1], L[0]이 0으로 초기화되며, reset 신호가 0일 때는 클럭 신호에 따라 L의 값이 순환하며 이동한다. 초기값 1000을 기준으로 0100, 0010, 0001로 순환하여 변화하는 것을 시뮬레이션을 통해 확인할 수 있었다. 이때 reset이 1인 경우 출력 값이 다시 0으로 초기화되는 것도 시뮬레이션 결과에서 명확히 나타났으며, 구현된 코드는 정확하게 링 카운터의 동작을 재현하였다.

UpDownCounter 모듈의 구현에서는 클럭 신호와 up 신호에 따라 카운터가 증가하거나 감소하는 동작을 확인하였다. 이 모듈은 clk, reset, up을 입력으로 받아 L[3]과 seg[6]을 출력으로 제공한다. 클럭 신호가 1인 경우, reset 신호가 1일 때는 L[3], L[2], L[1], L[0]과 seg 값이 모두 0으로 초기화되며, reset 신호가 0일 때는 up 신호에 따라 출력을 변화시킨다. up 신호가 1일 경우 출력 값은 클럭 신호에 따라 1씩 증가하며, seg 값은 U(0111110)를 나타낸다. 반대로 up 신호가 0일 경우 출력 값은 1씩 감소하며, seg 값은 d(1011110)을 나타낸다. 시뮬레이션 결과를 통해 클럭 신호의 활성화와 up 신호에 따라 L 값이 증가하거나 감소하고, seg 값이 성공적으로 출력되는 것을 확인할 수 있었다. 따라서, UpDownCounter 모듈 역시 요구 사항을 정확히 충족하며 구현되었음을 알 수 있었다.

종합적으로, 세 가지 모듈 모두 요구 사항에 맞게 설계되고 구현되었으며, 시뮬레이션 결과와 일치하는 동작을 보여 성공적으로 작동함을 확인할 수 있었다. 각 모듈의 동작은 클럭 신호와 입력 신호의 조합에 따라 정확히 변화하며, 초기화 신호에 따른 상태 초기화도 올바르게 수행되었다. Shift Register는 입력에 따라 데이터를 시프트하고, Ring Counter는 정해진 패턴으로 순환하며, UpDownCounter는 클럭 신호와 up 신호에 따라 카운트를 정확히 증가 및 감소시키는 동작을 구현하였다. 이러한 결과는 디지털 시스템 설계에서의 기본적인 모듈의 작동을 검증하며, 향후 복잡한 시스템의 구축에 있어서도 신뢰할 수 있는 기초를 제공한다.

**5. 추가 이론 조사 및 작성**

존슨 카운터(Johnson Counter)는 시프트 레지스터를 기반으로 한 순차 논리 회로의 일종으로, 링 카운터(Ring Counter)와 유사한 형태지만 출력이 더 다양하다. 존슨 카운터는 또한 트위스티드 링 카운터(Twisted Ring Counter)라고도 불리며, 디지털 회로에서 일정한 순환 패턴을 생성하는 데 사용된다.

예를 들어, 4비트 존슨 카운터의 경우, 다음과 같은 상태 순환을 가진다.

초기 상태: 0000

첫 번째 클럭 이후: 1000

두 번째 클럭 이후: 1100

세 번째 클럭 이후: 1110

네 번째 클럭 이후: 1111

다섯 번째 클럭 이후: 0111

여섯 번째 클럭 이후: 0011

일곱 번째 클럭 이후: 0001

여덟 번째 클럭 이후: 0000 (다시 초기 상태로)

존슨 카운터(Johnson Counter)는 n비트 시프트 레지스터와 반전된 출력 피드백으로 구성된 순차 논리 회로로, 트위스티드 링 카운터(Twisted Ring Counter)라고도 불린다. 기본적인 n비트 존슨 카운터는 n비트 시프트 레지스터를 사용하며, 각 플립플롭은 클럭 신호에 따라 동작한다. 이 시프트 레지스터의 최종 비트 출력은 반전되어 레지스터의 첫 번째 비트 입력으로 피드백된다. 동작 원리는 초기 상태에서 시프트 레지스터가 0이나 1의 특정 초기 값으로 설정되면서 시작된다. 클럭이 상승할 때마다 시프트 레지스터는 한 비트씩 오른쪽으로 시프트되며, 최종 비트의 반전된 값이 첫 번째 비트로 입력된다. 이 과정에서 레지스터는 n비트 길이의 특정 패턴을 순환하게 되고, n비트 존슨 카운터는 2n개의 서로 다른 상태를 거치며 순환한다.

존슨 카운터의 주요 특징은 상태가 2n개의 고유한 비트 패턴을 거치며 순환한다는 것이다. 이 순환 과정에서 절반의 시간 동안 0이 이동하고, 나머지 절반의 시간 동안 1이 이동하게 된다. 이는 링 카운터의 n개의 상태보다 두 배 많은 상태를 제공하여 더 복잡한 패턴을 생성할 수 있게 한다. 이러한 구조적 특징 덕분에 존슨 카운터는 n개의 플립플롭으로 2n개의 고유한 상태를 제공하여 링 카운터보다 더 많은 상태를 구현할 수 있는 효율적인 상태 수를 가진다. 또한, 다양한 순환 패턴을 제공하여 복잡한 타이밍 시퀀스와 제어를 쉽게 구현할 수 있다. 존슨 카운터는 시프트 레지스터와 반전 피드백이라는 간단한 구조로 구현 가능하여 설계가 간단하다.

존슨 카운터의 응용 분야로는 디지털 시퀀서, 타이밍 제어, 디지털 회로 테스트 등이 있다. 디지털 시퀀서에서는 특정 패턴의 순차 신호를 생성하는 데 사용되며, 타이밍 제어에서는 복잡한 타이밍 시퀀스를 생성하여 다양한 타이밍 제어 응용에 활용된다. 또한, 디지털 회로 테스트에서는 패턴 발생기나 디버깅 도구로서 다양한 상태 테스트를 지원한다. 이러한 특성 덕분에 존슨 카운터는 다양한 디지털 시스템에서 복잡한 상태 관리와 패턴 생성이 필요한 경우에 유용하게 사용된다.

