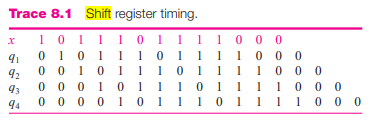
13주차 결과보고서

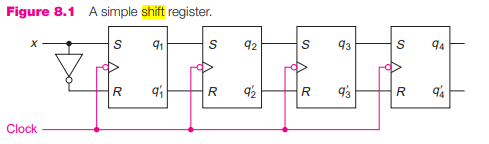
전공: 수학과 학년: 4학년 학번: 20171273 이름: 심현우

1. **4-bit shift register의 결과 및 simulation 과정에 대해서 설명하시오. (Verilog source, 출력 예시, 과정 상세히 적을 것!)**

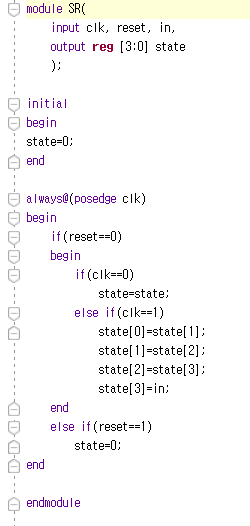
4-bit shift register는 clock pulse에 따라 기존 저장되어 있던 정보를 다른 정보로 바꾸어 저장한다. 4-bit이므로 저장된 정보는 4개이고 4개의 flip-flop을 사용한다. 이전 실습인 counter는 낮은 숫자부터 차례대로 증가하도록 설계가 되었지만 shift register는 한 bit씩 정보를 이동시키도록 설계되었다. 예를 들어, 1000의 정보가 저장되어 있을 때, 회로를 한 번 작동시키면 이후 저장되는 정보는 0100이 되도록 한다. 이를 timing table로 나타내면 다음과 같다.



왼쪽부터 순차적으로 clock pulse가 발생하고 x는 in으로 q1에 x의 값을 저장하는 역할을 한다. 따라서 timing결과를 봤을 때, x에 따라 q1이 저장되고 이후 clock pulse에 따라 q1값이 q2로, q2값이 q3로, q3값이 q4로 이동하는 것을 확인할 수 있다. 이를 회로로 구성하면 다음과 같다.

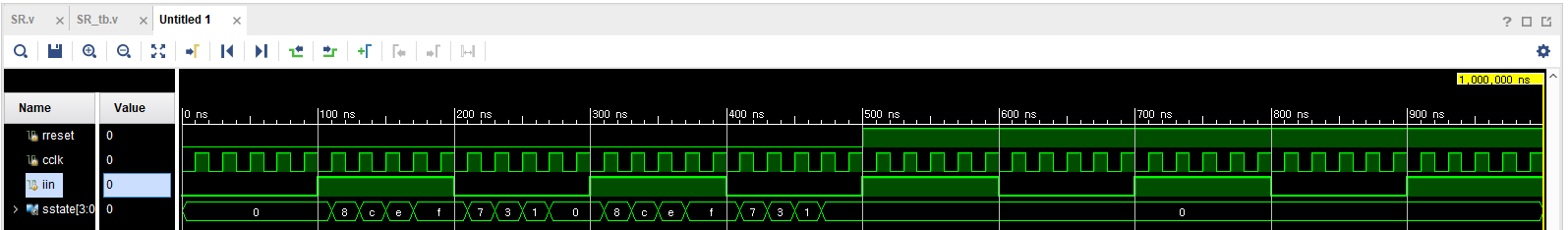


위에서 언급한 것처럼 4개의 flip-flop을 이용하여 구성하고 이전 flip-flop의 output이 다음 flip-flop의 input으로 들어간다. 이번 실습은 저번과 동일하게 직접 flip-flop을 assign으로 구성하지 않고, always구문을 이용하여 저장된 정보의 추이만 확인한다. 이를 위한 코드는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

Input으로는 clock pulse인 clk와 reset, in이 있고 output으로는 4-bit 정보가 있다. 정보는 이전 정보를 기억하도록 reg로 설정하고 reset은 정보를 0000으로 초기화하는 기능을 하며 in은 state[3]에 정보를 저장하는 역할을 한다. 초기 정보는 0000으로 초기화 했고, 각 clk의 posedge마다 작동을 지정했다. Reset이 0인 경우 clk=0일 때, 기존 정보를 유지하고, clk=1일 때, bit를 한 칸 씩 밀도록 했다. Reset이 1인 경우 정보를 0으로 초기화하도록 했다. Simulation 결과는 다음과 같다.



Simulation 결과를 보면 100ns부터 in의 값이 1이기 때문에 state[3]에 1의 값이 저장되어 최종 결과가 1000인 8이 된 것을 확인할 수 있다. 이후, 다시 state[3]에 1을 저장하고 나머지는 한 칸 씩 밀어 결과적으로 1100이 되고 c가 된 것을 알 수 있다. In이 0인 경우도 보자. 200ns부터 보면 저장된 정보가 7인 이후 clock pulse가 발생하게 되면 0111이었던 정보가 하나씩 밀려 0011이 되고 결과도 3이 된 것을 알 수 있다.

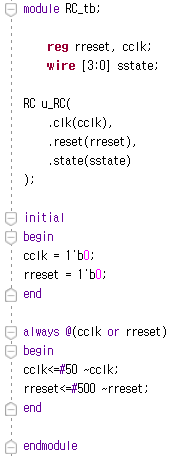
1. **4-bit Ring counter의 결과 및 simulation 과정에 대해서 설명하시오. (Verilog source, 출력 예시, 과정 상세히 적을 것!)**

Ring counter는 shift register의 기능을 하고 추가적으로 ring형태로 연결한 것이다. Shift register에서 마지막 데이터는 소멸되었는데 이를 다시 첫번째 bit로 옮겨 저장하는 것이 ring counter이다. 이를 표로 표현하면 다음과 같다.

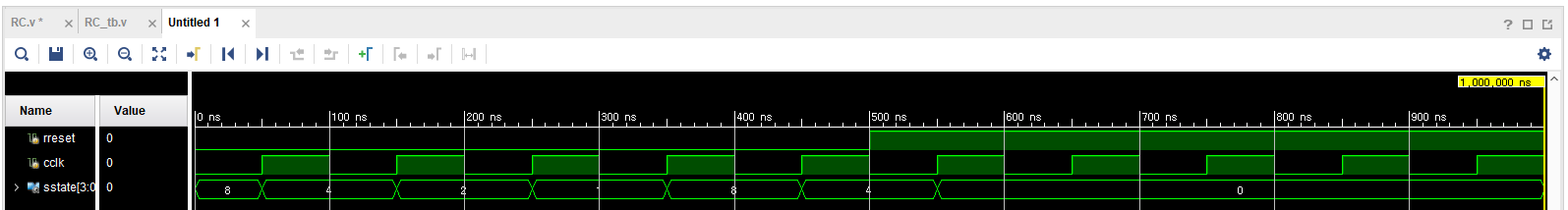
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Clock** | **Q0** | **Q1** | **Q2** | **Q3** |
| **0** | **1** | **0** | **0** | **00** |
| **1** | **0** | **1** | **0** | **0** |
| **2** | **0** | **0** | **1** | **0** |
| **3** | **0** | **0** | **0** | **1** |
| **4** | **1** | **0** | **0** | **0** |

마지막 4번 째 clock pulse에서 Q4의 1값이 소멸되지 않고 다시 Q0으로 저장된 것을 알 수 있다. Ring counter의 회로는 shift register와 같이 4개의 flip-flop으로 구성되며 처음 flip-flop의 output값이 다음 flip-flop의 input으로 들어가는 구조이다. 추가적으로 마지막 flip-flop의 output이 다시 처음 flip-flop의 input으로 들어간다. 이번 실습은 저번과 동일하게 직접 flip-flop을 assign으로 구성하지 않고, always구문을 이용하여 저장된 정보의 추이만 확인한다. 이를 위한 코드는 다음과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명 

Input으로는 clock pulse인 clk와 reset이 있고 output으로는 4-bit 정보가 있다. 정보는 이전 정보를 기억하도록 reg로 설정하고 reset은 정보를 0000으로 초기화하는 기능을 한다. 초기 정보는 1000으로 초기화 했고, 각 clk의 posedge마다 작동을 지정했다. Reset이 0인 경우 clk=0일 때, 기존 정보를 유지하고, clk=1일 때, bit를 한 칸 씩 밀도록 했다. 이 때, shift register와 다르게 마지막 데이터가 다시 처음 데이터로 넘어가야 하므로 =을 사용하면 blocking이 발생한다. 이는 처음 데이터와 마지막 데이터를 다루는데 순차적으로 진행하기 때문에 원하는 기능을 하지 못한다. 따라서 <=를 사용하여 non-blocking, 즉 begin, end안에 모든 작동이 동시에 진행하도록 한다. Reset이 1인 경우 정보를 0으로 초기화하도록 했다. Simulation 결과는 다음과 같다.



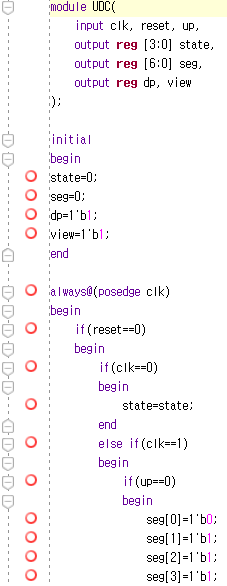
초기 정보는 8로 초기화 되어 있고 clock pulse가 발생할 때마다 4, 2, 1, 8로 변하는 것을 알 수 있다. 이는 1000->0100->0010->0001->1000->0100으로 순차적으로 변하는 것이고 ring counter의 기능을 잘 수행하는 것을 알 수 있다.

1. **4-bit UP/DOWN counter의 결과 및 simulation 과정에 대해서 설명하시오. (Verilog source, 출력 예시, 과정 상세히 적을 것!)**

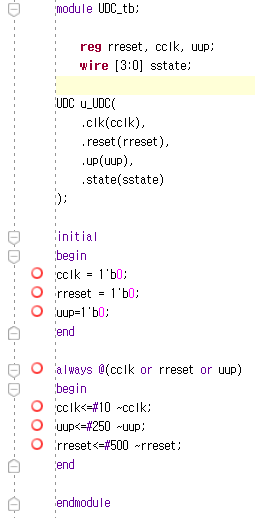
UP/DOWN counter는 지난 실습에 구현했던 counter에 기능을 추가한 것이다. Up/down을 선택하고, clock pulse가 발생했을 때, 1의 값을 증가시키거나 감소시킨다. 지난 실습에 counter는 Down이 없이 UP만 있는 counter와 같다. 지난 실습 때 구현한 2-bit binary counter를 기준으로 2-bit up/down counter의 동작을 표로 보면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Q0 | Q1 | X | Q0\* | Q1\* |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 |

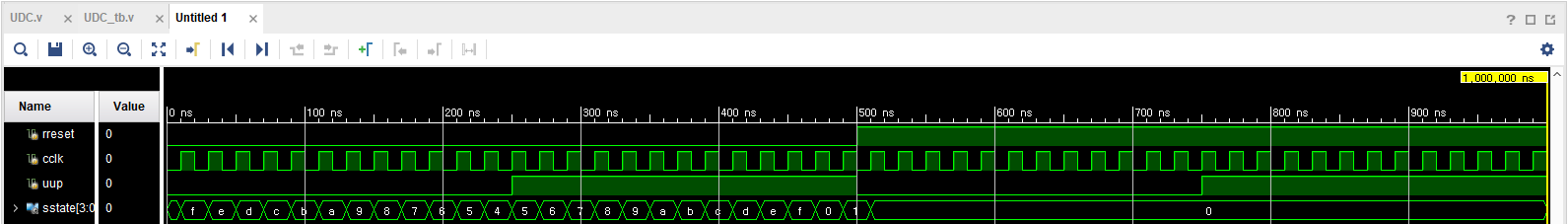
위의 표에서 X는 UP/DOWN을 선택하는 변수이다. X가 1이면 UP을, X가 0이면 DOWN의 동작을 수행한다. 기존 UP기능만 있는 counter는 Q0의 정보를 다음 flip-flop으로 input으로 사용했는데 DOWN의 기능을 추가하기 위해 ~Q0의 정보 또한 다음 flip-flop으로 보내야 한다. 이번 실습은 저번과 동일하게 직접 flip-flop을 assign으로 구성하지 않고, always구문을 이용하여 저장된 정보의 추이만 확인한다. 이를 위한 코드는 다음과 같다.

 텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명



위의 두개가 source코드이다. 이번 실습에서는 FPGA에서 segment에 U, D를 출력하기 위해 seg, dp, view 변수도 추가하였다. 그 외, clk와 reset이 있고, up인지 down인지 선택해줄 up변수, 4개의 bit정보를 저장하는 state변수를 추가했다. 초기 state는 0으로 초기화하고, 각 posedge clk마다 동작을 수행한다. Reset이 1이면 state의 값을 0으로 초기화한다. Reset이 0이면 clk=0일 때 정보를 유지하고 clk=1일 때, up/down을 선택한다. Up=1이면 up의 동작으로 state의 값을 1증가시키고 segment를 U로 표시하도록 한다. Up=0이면 down의 동작으로 state의 값을 1 감소시키고 segment를 D로 표시하도록 한다. Simulation 결과는 다음과 같다.



초기 state의 값은 0이고 up=0이므로 down 동작을 수행한다. 따라서 0000->1111->1110->1101->1100과 같이 순차적으로 감소한다. Up이 1로 세팅된 후 4였던 값은 순차적으로 5, 6, 7, 8, 9, a … 1씩 증가하는 것을 확인할 수 있다.

1. **결과 검토 및 논의사항**

FPGA로 pin을 연결해서 확인한 결과 shift register와 ring counter는 잘 작동하는 것을 쉽게 확인할 수 있었다. UP/DOWN counter는 segment를 연결하여 FPGA로 확인해야 하기 때문에 pin을 연결하는 과정에서 추가적인 작업이 필요했다. View 변수를 통해 어떤 digit에 표현할 지 연결하고 dp 또한 1로 설정하여 segment밑 dp에 해당하는 점에 점등이 잘 되고 있는 것을 확인할 수 있도록 연결했다. Segment변수 7개는 각각 segment0부터 segment6까지 연결하여 up인 경우 U, down인 경우 D를 표시할 수 있도록 했다. FPGA로 동작을 확인한 결과 up으로 설정하면 U가 잘 점등되는 것을 확인할 수 있었고 counter동작 또한 up으로 잘 작동하는 것을 알 수 있었다. Down의 경우도 잘 작동하는 것을 알 수 있었다.

1. **추가 이론 조사 및 작성**

source코드를 작성함에 있어 <=과 =을 잘 구분하여 사용하여야 한다. 특히 이번 ring counter를 구현하는데 있어 =을 사용했을 때, 동작이 잘 구현되지 않는 것을 확인할 수 있다. 이는 c언어로 코딩할 때, 변수의 값을 바꾸는 swap의 함수를 구현할 때의 문제와 같다. 이때 swap의 함수를 구현하려면 tmp변수를 추가하여 중간에 임시로 tmp 변수에 정보를 저장하여 swap을 진행해야 한다. Ring counter 또한 tmp 변수가 없으면 중간에 임시적으로 정보를 저장하지 못하고 제대로 작동하지 않는다. verilog에서는 따로 tmp 변수를 추가할 수 없기 때문에 모든 동작을 한번에 수행할 수 있도록 non-blocking인 <=을 사용하여 회로를 구현해야 한다.