1. 11주차 결과보고서

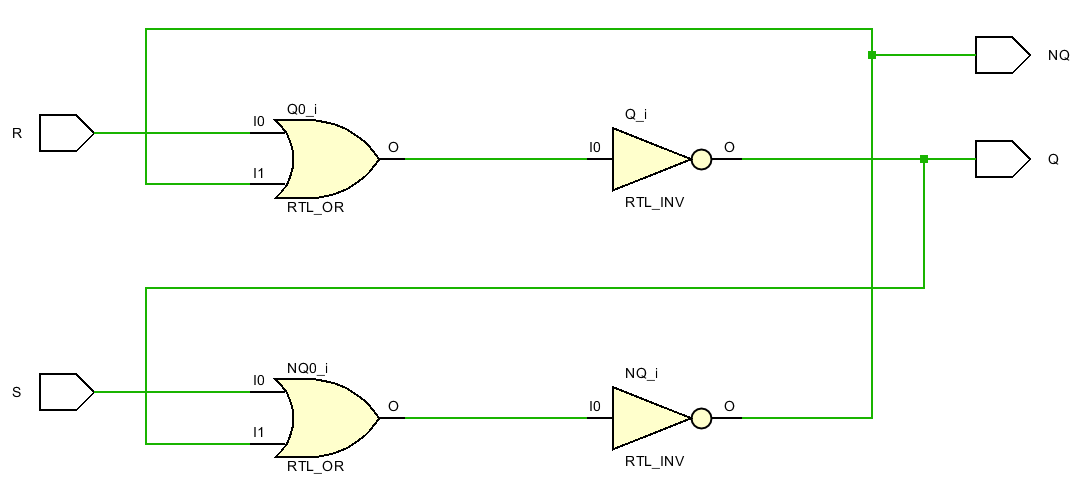
|  |
| --- |
| - RS latch  - RS flip flop  - D latch  - D flip flop  - jk flip flop  - 결과 검토 및 논의 사항  - 추가 이론 |

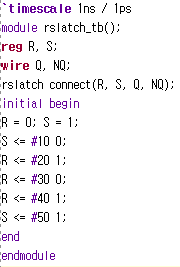
**20141196 김성희**

1. RS latch

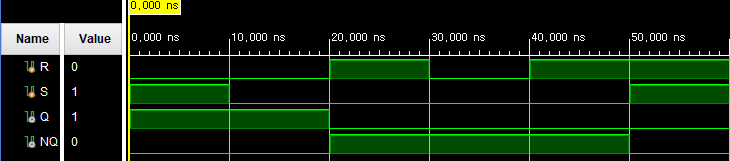
|  |  |
| --- | --- |
| **-논리식** Q\* = !(R | NQ) NQ\* = !(S | Q)  Q\*, NQ\*는 각각 next Q, NQ를 뜻한다. | **-코드**  nor(A, B, C)는 A=!(B | C)를 의미한다. NQ = !Q를 의미한다. |

**-회로도(schematic)**

  
**-simulation**

 <- simulation code

Initial begin end 문에서 RS의 input을 순서대로 변환 시킨 것을 볼 수 있다. RS는 10ns 간격으로 01, 00, 10, 00, 10, 11로 바뀐다.

R, S는 각각 Reset, Set을 의미한다. 따라서 R이 1이면 Q(NQ)가 0(1)이 되고, S가 1이면 Q(NQ)가 1(0)이 된다. 따라서 RS가 01이면 Q NQ가 10이고 10이면 01이다. RS가 00이면 이전 값이 보존되는 것을 알 수 있다. 그러나 RS가 11인 경우 Q NQ가 00이 되며 이는 아무 의미가 없는 값이다. 따라서 RS latch의 input으로 11은 사용하지 않는다.

**-truth table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | Output | |
| 입력 순서 | R | S | Q | NQ(or ~Q) |
| (1) | 0 | 1 | 1 | 0 |
| (2) | 0 | 0 | 1 | 0 |
| (3) | 1 | 0 | 0 | 1 |
| (4) | 0 | 0 | 0 | 1 |
| (5) | 1 | 0 | 0 | 1 |
| (6) | 1 | 1 | 0 | 0 |

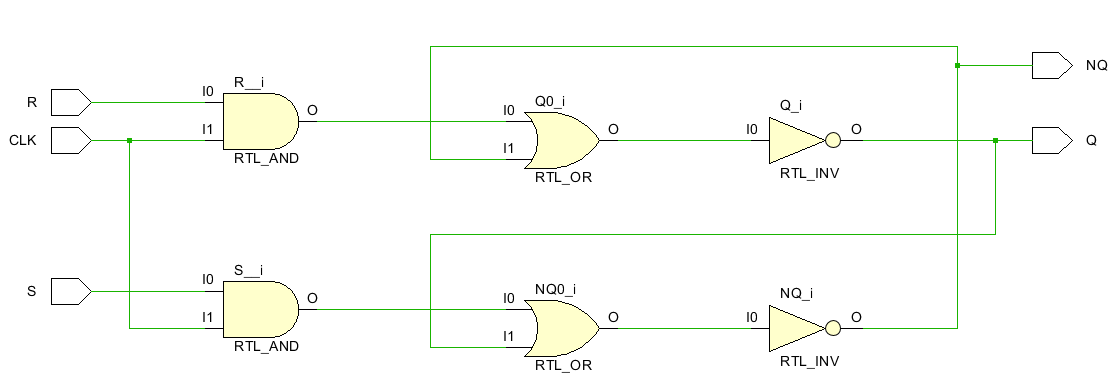
(1)->(2)와 (3)->(4)를 보면 00일 때 이전 값이 보존되는 것을 볼 수 있다.  
(6)을 보면 Q, NQ가 00이어서 의미가 없는 값임을 볼 수 있다.

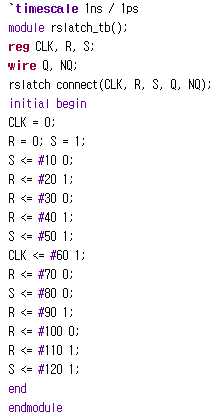
1. RS flip flop
   1. **가. Gated RS latch (= RS flip flop)**

**1. NOR 형식**

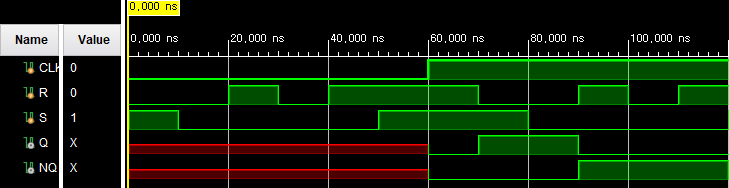
|  |  |
| --- | --- |
| **-논리식** Q\* = !( (R & CLK) | NQ) NQ\* = !( (S & CLK) | Q)  Q\*, NQ\*는 각각 next Q, NQ를 뜻한다. | **-코드**  and(A, B, C)는 A=B&C를, nor(A, B, C)는 A=!(B | C)를 의미한다. NQ = !Q를 의미한다. |

**-회로도(schematic)**

  
**-simulation**

 <- simulation code

Initial begin end 문에서 RS의 input을 순서대로 변환시킨 것을 볼 수 있다. RS는 10ns 간격으로 01, 00, 10, 00, 10, 11로 바뀐다. CLK = 0일 때와 CLK = 1일 때의 input 순서가 똑같다. CLK = 0 이전에 RS는 정해지지 않았다.

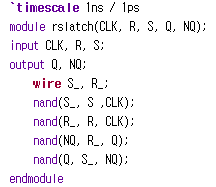


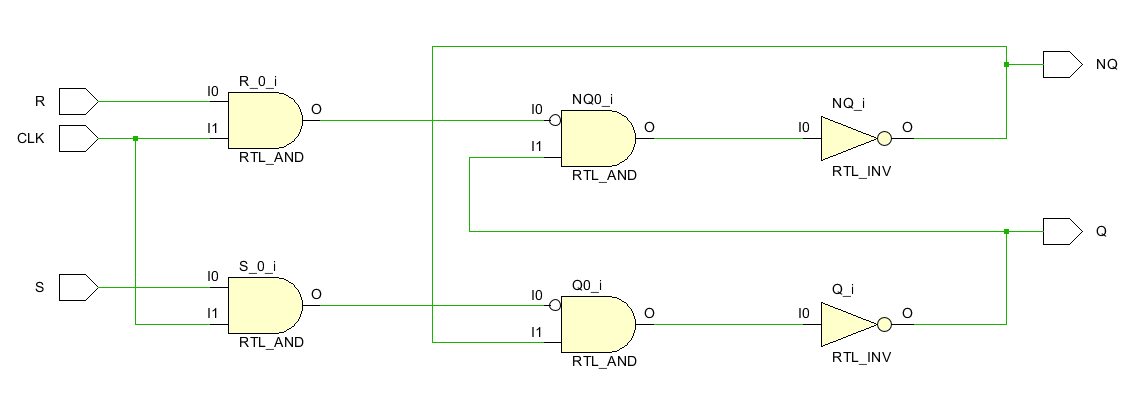
CLK=0이면 NOR게이트 input RS가 00임을 RS latch 모듈 코드를 통해 알 수 있다. 그리고 RS latch 실험을 통해 input값이 00이면 이전 값이 보존되는 것을 알 수 있다. 그러나 gated rs latch 실험에서 사용한 시뮬레이션 모듈 코드에는 CLK=0 이전에 RS값을 정의하지 않았기에 그림에서 빨간색 선으로 표시된다. (값을 모른다는 의미이다.) CLK = 1이면 RS latch의 simulation 그림과 같은 것을 볼 수 있다. 따라서 latch의 기능을 CLK로 enable 시킬 수 있다.

**-truth table**

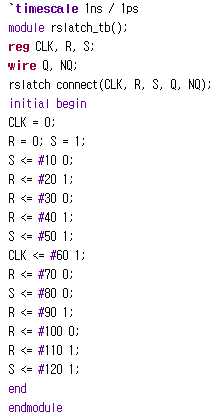
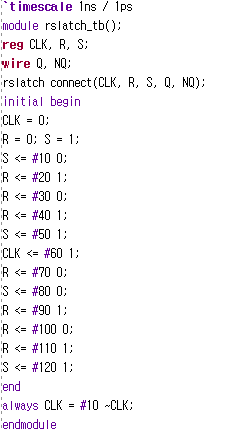
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | Output | |
| 입력 순서 | CLK | R | S | Q | NQ(or ~Q) |
| (1) | 0 | 0 | 1 | ? | ? |
| (2) | 0 | 0 | ? | ? |
| (3) | 1 | 0 | ? | ? |
| (4) | 0 | 0 | ? | ? |
| (5) | 1 | 0 | ? | ? |
| (6) | 1 | 1 | ? | ? |
| (1) | 1 | 1 | 1 | 0 | 0 |
| (2) | 0 | 1 | 1 | 0 |
| (3) | 0 | 0 | 1 | 0 |
| (4) | 1 | 0 | 0 | 1 |
| (5) | 0 | 0 | 0 | 1 |
| (6) | 1 | 0 | 0 | 1 |

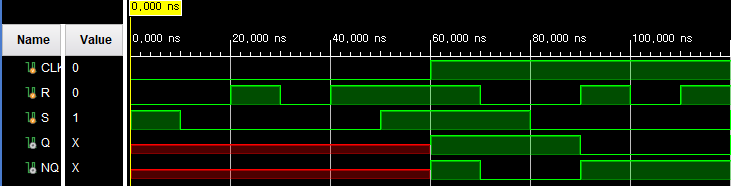
**2. NAND 형식**

**-코드**S와 R값을 CLK와 nand 연산후에 그 결과값을 wire S\_, R\_를 통해 Q와 NQ값을 구하기 위한 nand 연산의 입력으로 연결해준다.

**-회로도(schematic)**

**-simulation**

****NOR의 simulation code와 동일



결과는 nor형식과 동일하다. 단 RS가 11일 때 nor 형시에서는 Q NQ가 00이었으나 nand형식에서는 11임을 볼 수 있는데 의미가 없는 입력이므로 동일하다고 봐도 무방하다.

**-truth table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | Output | |
| 입력 순서 | CLK | R | S | Q | NQ(or ~Q) |
| (1) | 0 | 0 | 1 | ? | ? |
| (2) | 0 | 0 | ? | ? |
| (3) | 1 | 0 | ? | ? |
| (4) | 0 | 0 | ? | ? |
| (5) | 1 | 0 | ? | ? |
| (6) | 1 | 1 | ? | ? |
| (1) | 1 | 1 | 1 | 1 | 1 |
| (2) | 0 | 1 | 1 | 0 |
| (3) | 0 | 0 | 1 | 0 |
| (4) | 1 | 0 | 0 | 1 |
| (5) | 0 | 0 | 0 | 1 |
| (6) | 1 | 0 | 0 | 1 |

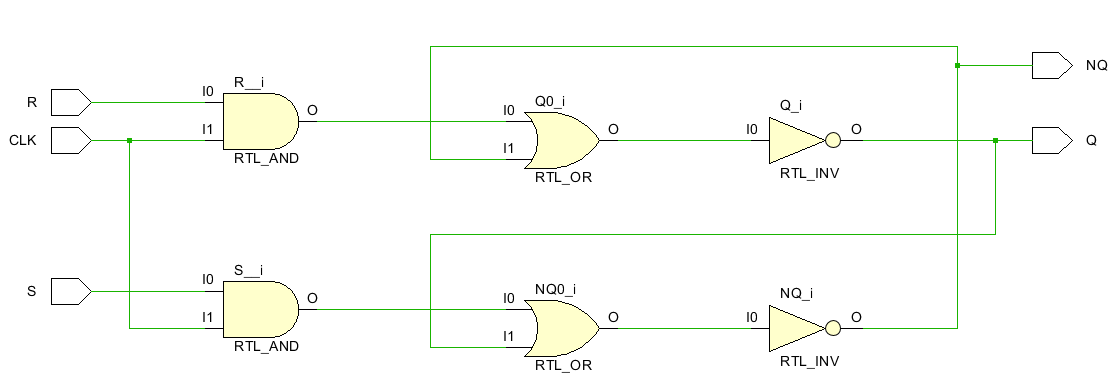
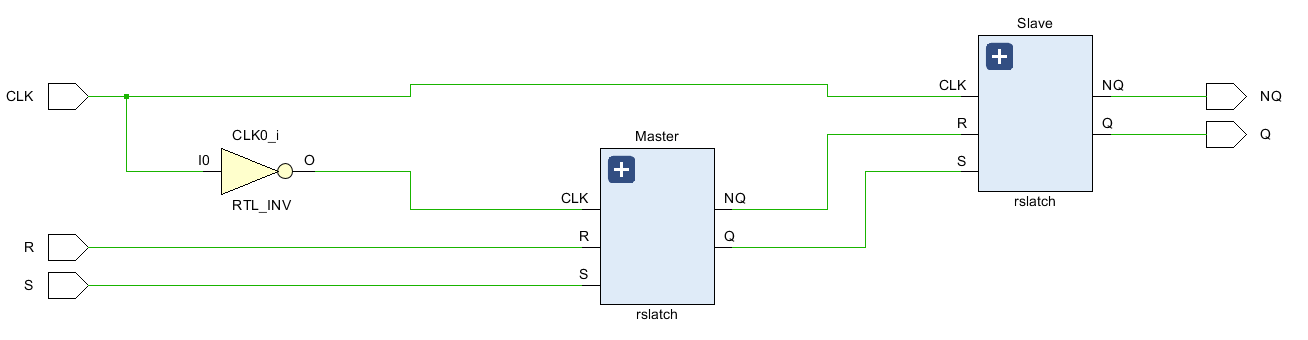
nor형식과 동일하다.

* 1. **나. RS flip flop (leading edge triggerd)**

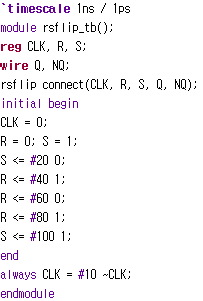
**1. nor형식**

|  |  |
| --- | --- |
| **-코드** |  |

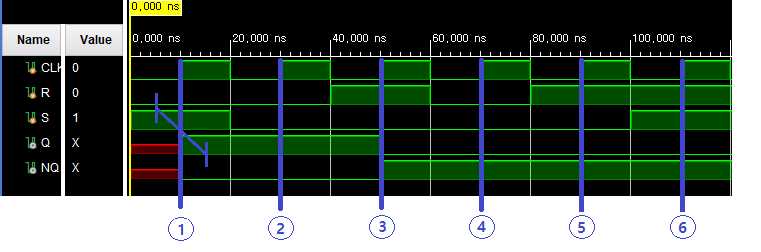
**-회로도(schematic)**

 ****첫번째는 Master, Slave 블랙박스 내부 회로도고, 두번째는 RS Flip Flop 회로도다.  
CLK가 0일 때, Master 회로의 결과값이 바뀌고 CLK가 1일 때 Slave 결과값이 바뀐다. 즉 CLK가 0에서 1로 rising할 때 결과값이 변경된다.

**-simulation**

 <- simulation code

Initial begin end 문에서 RS의 input을 순서대로 변환시킨 것을 볼 수 있다.

****  
1, 2, … , 6처럼 CLK가 0에서 1로 rising할 때 결과값이 바뀐다. 1의 경우 rising하는 그 시각에 RS가 01이기 때문에 결과값 Q NQ가 10으로 변경된 것을 볼 수 있다. 2와 3사이에 RS가 00에서 10으로 바뀜에도 Q NQ가 01이 아닌 10을 계속 유지하는 것을 통해 오로지 CLK의 rising이 일어날 경우에만 값이 바뀌는 것을 알 수 있다. 그 외에 RS값에 따라 Q 와 NQ가 어떻게 변하는지는 RS latch와 동일 하게 작용하는 것을 볼 수 있다. (6번의 경우는 의미 없는 값이므로 무시)

**-truth table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Input | | | | Output | |
| 입력 순서 | CLK | R | S | Q | NQ(or ~Q) |
| (1) | 0->1 | 0 | 1 | 1 | 0 |
| (2) | 0 | 0 | 1 | 0 |
| (3) | 1 | 0 | 0 | 1 |
| (4) | 0 | 0 | 0 | 1 |
| (5) | 1 | 0 | 0 | 1 |
| (6) | 1 | 1 | 0 | 1 |

RS가 11인 경우는 의미없는 값이므로 무시

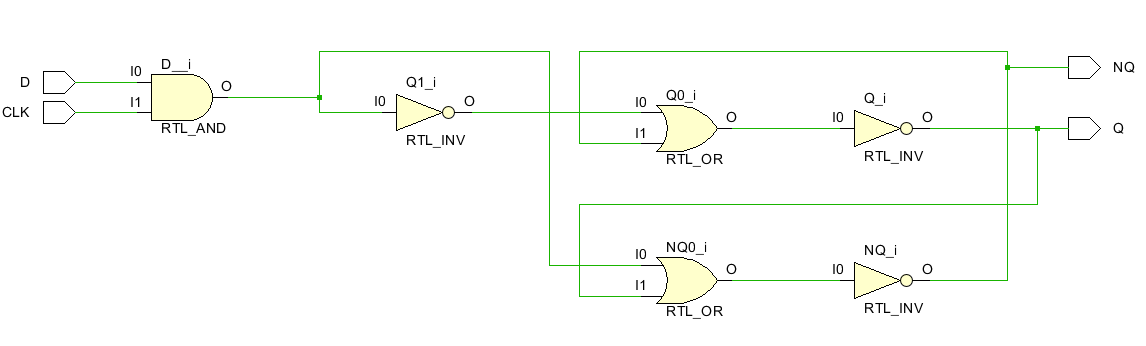
**2. NAND 형식**

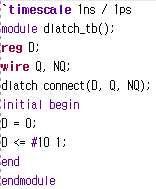
NAND gated RS latch 모듈 두개를 사용하는 것 빼고는 flip flop 코드와 simulation 코드, simulation이 모두 NOR형식과 동일하다.

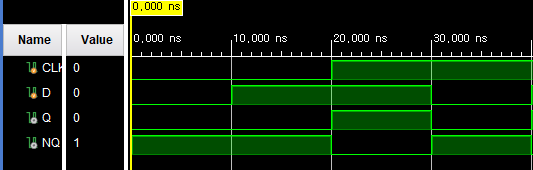
1. D latch

|  |
| --- |
| **-코드**  nor(Q,D,NQ) : Q = !(D | NQ) |

**-회로도(schematic)**

**-simulation**

 <- simulation code

  
D=0 -> Q=0, D=1 -> Q=1, 즉 D=1이면 Set을, D=0이면 Reset을 의미한다.  
(CLK = 0이면 항상 Q = 0)

**-truth table**

|  |  |  |
| --- | --- | --- |
| Input | Output | |
| D | Q | NQ(or ~Q) |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

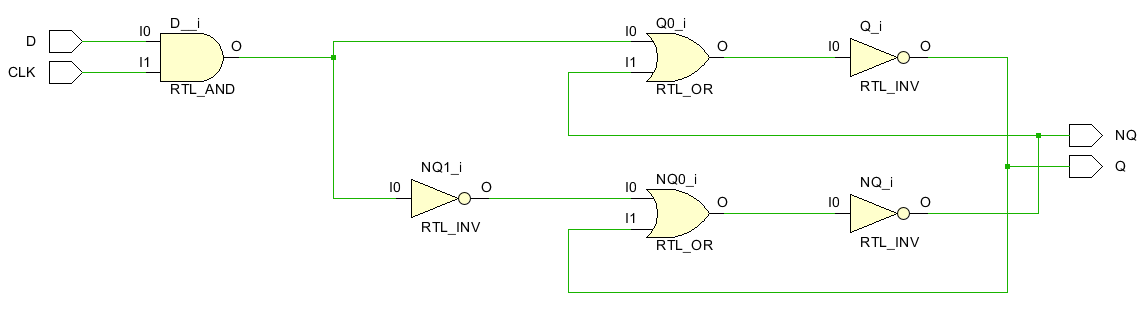
1. D flip flop

가. gated D latch

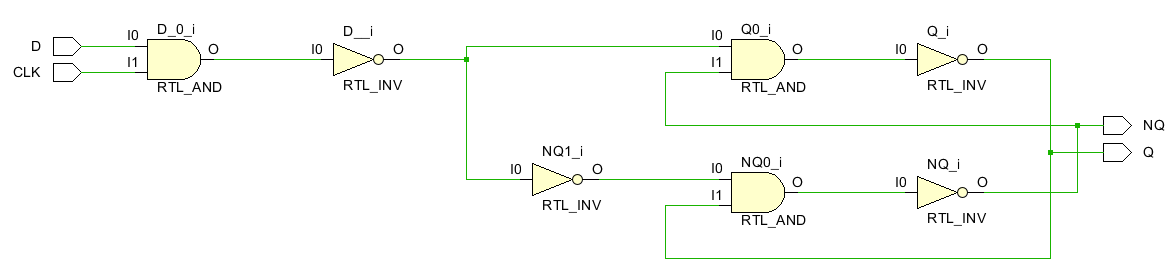
|  |  |
| --- | --- |
| **-코드 (NOR)**  D\_ = D AND CLK Q = D\_ NOR NQ NQ = !D\_ NOR | **(NAND)** |

**-회로도(schematic)**

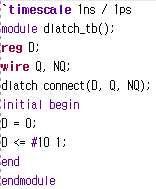
**NOR**

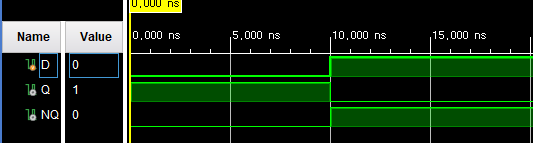


**NAND**

****

**-simulation**

 <- simulation code

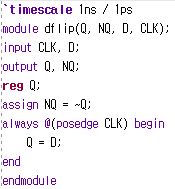
  
D=0 -> Q=0, D=1 -> Q=1

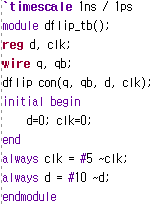
**-truth table**

|  |  |  |
| --- | --- | --- |
| Input | Output | |
| D | Q | NQ(or ~Q) |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

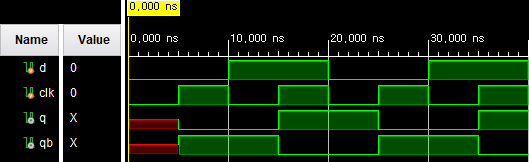
나. D flip flop (edge trigger)

**-코드**

@(posedge CLK) 코드를 통해 edge trigger를 구현했으며 assign NQ = ~Q;를 통해 NQ값을 설정했다. Q는 D의 값과 동일 하기 때문에 Q = D를 통해 간단하게 D flip flop을 구현했다.

**-시뮬레이션**

시뮬레이션 코드다. 초기 설정을 d=0, clk=0으로 맞춘뒤 d값(입력값)은 10ns마다, clk는 10ns마다 바뀌도록 설정했다.



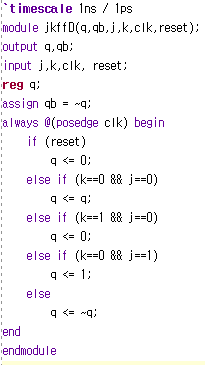
빨간 부분은 초기값이 정해지지 않았기 때문이다. d가 0으로 설정되어 있지만 rising edge가 되기 전이므로 5ns까지는 q값이 정해지지 않는다. 5ns 이후에는 rising edge마다 q값이 d값과 동일하게 변경되는 것을 볼 수 있다.

**-truth table**

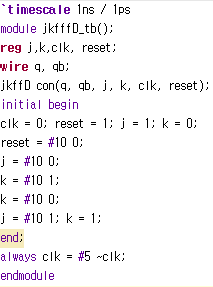
|  |  |  |
| --- | --- | --- |
| Input | Output | |
| D | Q | NQ(or ~Q) |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

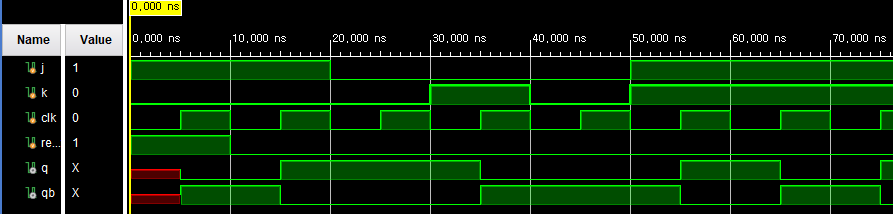
1. JK flip flop

**-코드 (rising edge triggered flip flop)**

 Reset이 1인 경우 q와 qb값을 설정할 수 있도록 구현했다. 그 이유는 처음 결과 값을 구할 때 q와 qb의 결과를 구하지도 않고 사용해야 하는 경우가 있기 때문이다. 즉 해당 모듈을(플립플롭을) 사용하기 위해서 먼제 reset 1일 통해 q값을 0으로 설정해야 한다. 나머지 else if 와 else에 있는 코드는 각각 jk가 00 01, 10, 11일 때 q가 어떻게 바뀌는지 구현한 코드다. 그리고 clk가 0에서 1로 바뀔 때 q값이 바뀌도록 @(posedge clk)를 구현했다. 즉 해당 모듈은 rising(leading) edge triggered JK flip flop을 구현한 코드다.

**-simulation**

왼쪽 코드는 시뮬레이션 코드, 즉 테스트벤치 코드다. Clk를 5ns마다 변화시켰고 처음 q값을 설정해주기 위해 reset에 1을 넣어준 뒤 10ns 후에 reset에 0을 넣어서 jkffD 모듈이 j,k값에 따라 q값이 바뀌도록 설정했다. Jk의 입력값은 순서대로 10 00 01 00 11 11 11….이다.



위 시뮬레이션 코드를 시뮬레이션 했을 때의 결과다. 0ns 부터 10ns까지 reset에 1을 넣어주어 5ns의 rising edge에서 q값이 0으로 설정되도록 하는 것을 볼 수 있다. 이어서 jk가 10으로 바뀌면서 q에 1이 할당 되고 00일 때는 q값을 보존, 다시 01이 되면서 q값이 0으로 바뀌는 것을 볼 수 있다. 50ns부터 jk에 11이 할당 되면서 rising edge 마다 q값이 반전되는 것도 볼 수 있다.

**-truth table (위 시뮬레이션에 대한 진리표)**

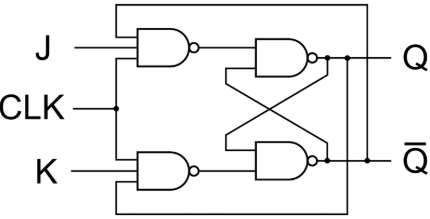
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 순서 | reset | j | k | q |
| (1) | 1 | X | X | 0(reset) |
| (2) | 0 | 1 | 0 | 1(set) |
| (3) | 0 | 0 | 0 | 1(보존) |
| (4) | 0 | 0 | 1 | 0(reset) |
| (5) | 0 | 1 | 1 | 1(toggle) |
| (6) | 0 | 1 | 1 | 0(toggle) |

기타 논의 및 검토 사항 + 추가 이론

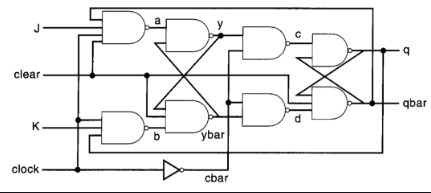
- latch와 gated latch와 flipflop의 차이는 각각 (1)clock이 없다. (2)clock이 있다. (3)clock이있고 edge trigger가 있다. 로 구분할 수 있다. 그러나 종종 gated latch와 플립플롭을 혼합해서 사용하는 것으로 보아 용어의 구분보다는 기능의 구분으로 분류하고 기억하는 것이 좋아 보인다.

- JK flip flop의 경우 gate level modeling에 실패했다. 그림1의 경우는 JK가11일 때 토글이 먹히지 않았고, 그림2의 경우 중간 결과값인 y, ybar를 강제로 설정할 수가 없어서 먹히질 않았다.

<그림1>



<그림2>



따라서 behavioral modeling을 통해 rising edge trigger jk flip flop을 구현했다.

- JK flip flop의 경우 rs flip flop의 확장판 버전이라는 느낌을 받았다. JK flip flop처럼 RS flip flop의 구조를 이용하는 flip flop 중에 T flip flop이 존재하는데 구현법은 아래 회로도를 참고하면 알 수 있다.

