컴퓨터 공학 기초 실험2 보고서

실험제목:

Latch & flip-flop design with/without reset/set

Traffic Light Controller with/without Left Turn Signals

실험일자: 2014년 09월 29일 (목)

제출일자: 2014년 10월 06일 (목)

학 과: 컴퓨터공학과

담당교수: 이혁준 교수님

실습분반: 목요일 0, 1, 2

학 번: 2013722095

성 명: 최 재 은

1. 제목 및 목적
   1. 제목

Latch & flip-flop design with/without reset/set

Traffic Light Controller with/without Left Turn Signals

* 1. 목적

srLatch의 원리를 이용하고 이를 적용하여 D-latch와 D-filp flop을 설계해 본다. 또 이를 이를 활용해 bit data를 기억할 수 있도록 설계한다. Reset신호에 의해서 기억하고 있는 데이터를 초기화 시키는 dff\_reset을 설계해 보고 이를 활용하여 신호등 시스템을 간략하게 설계해 본다.

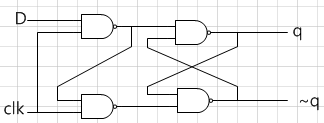
1. 원리(배경지식)

Latch란 클럭을 받지 않는 형태의 비동기식 기억소자로써 SR latch의 경우 Set / Reset을

입력으로 받아 reset이 0일 경우에만 s값을 출력해주는 모듈입니다. D latch는 SR latch와

비슷하게 저장된 비트를 유지하거나 투과시키는 역할을 해줍니다. 해당 실험에서는 clk을 받아서 이에 따라 값을 기억하거나 투과시켜줍니다. DFF는 인버터를 통해서 클럭의 반전된 값을 갖는데 이를 통해 각각 출력되는 값을 조절해줍니다. 이전의 상태를 계속 유지하는 성질을 갖고 있기 때문에 클럭을 통해서 이를 내보낼지 계속 기억하고 있을지를

판단하게 됩니다. 이때 입력값은 클럭의 엣지에서 받아들이며 엣지가 발생하지 않을 때는 출력이 변하지 않고 유지됩니다. DFF\_reset은 추가적으로 reset 인자를 입력으로 받아서 and gate를 통하게 되는데 reset이 0이면 출력이 없다가 1로 바뀌면서 입/출력을 산출합니다. 4개의 nand gate를 이용하여 D-latch를 구현하는 방법으로는 1번 nand gate에 d입력과 clk값을 입력으로 주고 이의 출력을 다른 nand gate의 입력으로 넣습니다. 이때의 nand gate에는 clk가 입력으로 같이 들어가는데 지금까지의 두 nand gate의 출력을 nand gate 두 개를 이용하여 구성한 ~SR latch의 입력으로 넣어주면 D latch가 완성이 됩니다. 그림으로 표현하면 다음과 같습니다.



실습에서 구현한 enabled Dff는 module을 instance하여 해당 조건에 맞도록 자체 연산하여 출력하는 방식이었지만 이를 synchronous set/reset으로 구현하게 되면 always문에서 if의 조건에 따라 값이 바로 출력되도록 구현할 수 있습니다.

1. 설계 세부사항

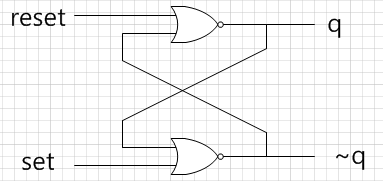
1) SR latch

- nor gate 두 개를 이용하여 설계하였습니다.

- 첫 번째 nor gate는 reset 신호와 두 번째 nor gate의 output인 ~q를 입력으로 받습니다.

- 두 번째 nor gate는 set 신호와 첫 번째 nor gate의 output 인 q를 입력으로 받습니다.

- 다음과 같은 형태로 표현됩니다.



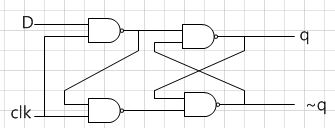
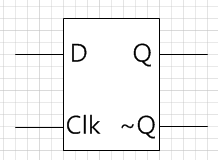
- 다음과 같은 동작을 합니다.

|  |  |  |  |
| --- | --- | --- | --- |
| Input | | Output | |
| S | R | Q | ~Q |
| 0 | 0 | Q(prev) | ~Q(prev) |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

2) D latch

- clock이 enable 상태를 유지하는 동안에 입력 값인 d의 변화를 출력합니다.

- 다음과 같은 형태로 표현됩니다.

- 다음과 같은 동작을 합니다.

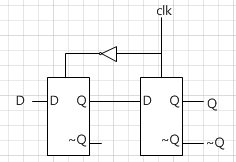
|  |  |
| --- | --- |
| CLK | Q |
| 0 | Q(prev) |
| 1 | D |

3) D flip flop

- clock의 rising edge에서만 D 값으로 출력이 바뀌게 되는데 다른 경우에는 D값이 바뀌더라고 이전에 기억하던 값을 그대로 유지합니다.

- 처음 D latch에는 ~clk를 clk으로 넣어주고 D를 입력으로 받습니다. 이때 발생한 Q를 다음 D latch의 D입력으로 넣어줍니다. 이 때의 D latch는 clk를 clk으로 받습니다.

- 다음과 같은 형태로 표현됩니다.



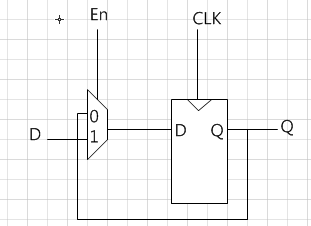
- 다음과 같은 동작을 합니다.

|  |  |
| --- | --- |
| CLK | Q |
| Rising Edge | D |
| in other case | Q(prev) |

4) enabled D flip flop

- D flip flop 앞에 1bit Multiplexer를 추가하여 입력 data의 사용 여부를 결정합니다.

- 다음과 같은 형태로 표현됩니다.



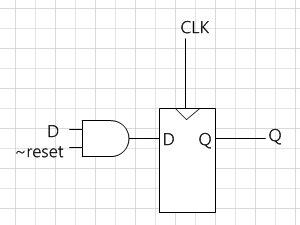
- 다음과 같은 동작을 합니다.

|  |  |  |
| --- | --- | --- |
| En | CLK | Q |
| 0 | Rising edge | Q(prev) |
| In other case | Q(prev) |
| 1 | Rising edge | D |
| In other case | Q(prev) |

5) resettable D flip flop

- D flip flop앞에 and gate를 더해 reset bit를 통해서 입력 값의 data 사용여하를 결정합니다.

- 다음과 같은 형태로 표현됩니다.



- 다음과 같은 동작을 수행합니다.

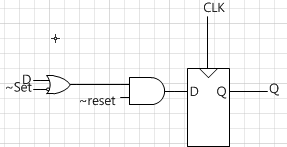
|  |  |  |
| --- | --- | --- |
| ~reset | CLK | Q |
| 0 | Rising edge | Q(prev) |
| In other case | Q(prev) |
| 1 | Rising edge | D |
| In other case | Q(prev) |

6) set/resettable D flip flop

- or gate의 input으로 D와 set이 들어가게 되고 이의 출력과 ~reset을 and gate의 입력으로 설계합니다. 이 때의 출력이 flip flop의 입력으로 들어가게 됩니다.

- set/ reset은 active low에 동작합니다.

- 다음과 같은 형태로 표현됩니다.



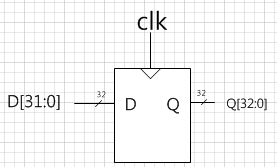
- 다음의 동작을 수행합니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Input | | | | Output |
| ~R | S | D | CLK | Q |
| 0 | X | X | X | 0 |
| 1 | 0 | x | X | 1 |
| 1 | 1 | 0 | Rising edge | 0 |
| 1 | 1 | 1 | Rising edge | 1 |
| 1 | 1 | X | Falling edge or 0 or 1 | Q(prev) |

7) 32bits register

- 32개의 flip flop을 한줄로 늘여놓는 형식으로 구현함으로써 32bit를 기억합니다.

- 다음과 같이 표현될 수 있습니다.



8) asynchronous/synchronous set/resettable d flip flop

- always 문을 사용하여 주기를 주고 if 문을 통해서 Q의 값을 지정해 주는 방식으로

구현되었습니다.

- asynchronous flip flop은 clk에 따라서 연산이 진행되지만 set값과 reset값에 따라서 출력이 다르게 됩니다.

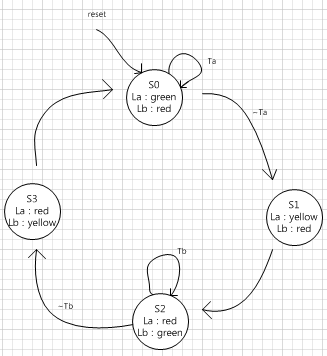
- synchronous flip flop은clk에 따라서만 연산이 진행됩니다.

9) Traffic Light Controller without Left Turn signals

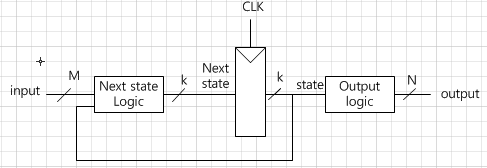
- traffic light는 Green(00), Yellow(01), Red(10)순으로 변합니다.

- La가 00, 01인 경우 Lb는 10을 유지합니다. 반대의 경우 역시 동일합니다.

- 다음과 같은 순서로 진행됩니다.



- 실제 모듈 구성은 다음과 같습니다.



**<State Transition Table>**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Current State** | | **Inputs** | | **Next State** | |
| **Q1** | **Q0** | **Ta** | **Tb** | **D1** | **D0** |
| **0** | **0** | **0** | **X** | **0** | **1** |
| **0** | **0** | **1** | **X** | **0** | **0** |
| **0** | **1** | **X** | **X** | **1** | **0** |
| **1** | **0** | **X** | **0** | **1** | **1** |
| **1** | **0** | **X** | **1** | **1** | **0** |
| **1** | **1** | **x** | **X** | **0** | **0** |

- 위의 진리표를 정리하면 다음과 같습니다.

D1 = Q1 (+) Q0

D0 = (~Q1 \* ~Q0 \* ~Ta) + (Q1 \* ~Q0 \* ~Tb)

**<Output table>**

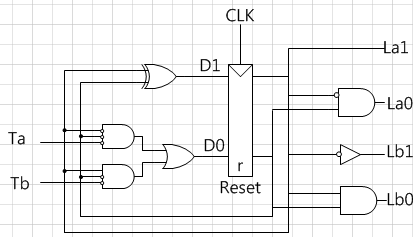
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Current state | | Output | | | |
| Q1 | Q0 | La1 | La0 | Lb1 | Lb0 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |

- Light A/B의 진리표는 위와 같으며 이를 정리하면 다음과 같습니다.

La1 = Q1 / La0 = ~Q1 \* Q0

Lb1 = ~Q1 / Lb0 = Q1 \* Q0

- 전체 module 구성은 다음과 같고 아래와 같이 표현됩니다.



**<Module configuration>**

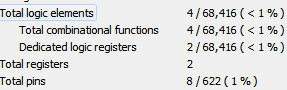
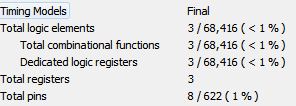
|  |  |  |
| --- | --- | --- |
| 구분 | 이름 | 설명 |
| Top module | tl\_cntr | Traffic light controller의 top module |
| Sub module | ns\_logic | Traffic light controller의 next state를 결정하는  combinational logic |
| \_register2\_r | 2bit resettable register with active low asynchronous  Reset module (내부에 \_dff\_r\_async를 instance)  - 현재 state의 값을 저장하고 있다. |
| \_dff\_r\_async | Resettable D flip-flop with active low asynchronous  reset |
| o\_logic | 현재 state의 값에 기반하여 output 값을 결정하는 combinational logic |

**-** 모듈 인자의 구성은 다음과 같습니다.

**<Traffic light controller>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Module  이름 | 구분 | 이름 | 비트 수  (bit) | 설명 |
| tl\_cntr | Input | Clk | 1 | Clock |
| reset\_n | 1 | Active low동작하는 reset 신호로  State를 초기화함 |
| Ta | 1 | Traffic sensor A |
| Tb | 1 | Traffic sensor B |
| Output | La | 2 | 신호등 값 출력 A |
| Lb | 2 | 신호등 값 출력 B |

- 해당 프로그램을 combinational logic / behavior design으로 구현할 경우에 대해 각각

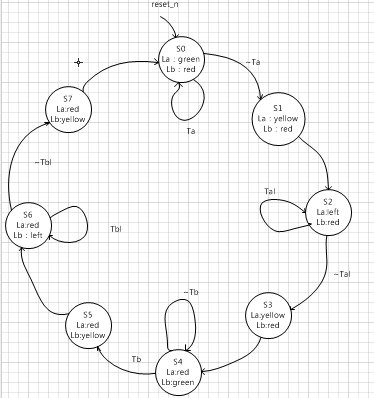
다음과 같은 결과를 보이는데 behavior design의 경우에 logic의 감소한 것을 확인할 수 있습니다. 다만 이 프로그램의 size가 별로 크지 않기 때문에 차이가 적을 뿐이고 프로그램 사이즈가 커질수록 그 차이가 극명하게 나타날 것이라고 생각됩니다.

10) Traffic Light Controller with Left Turn signals

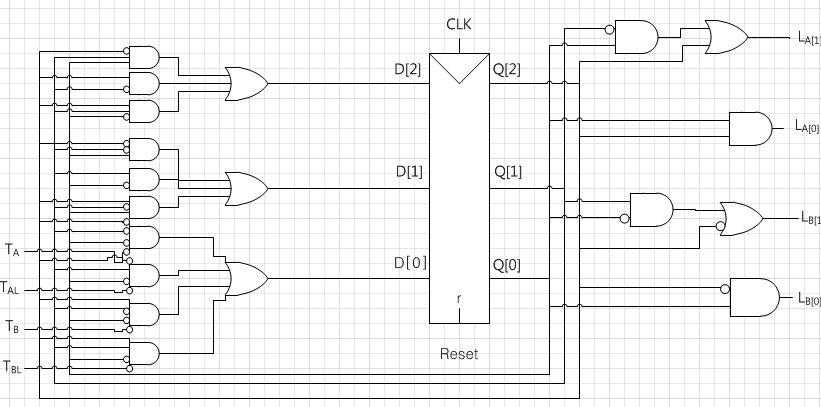
- Traffic signal은 Green(00), Yellow(01), Left(10), Yellow(01), Red(11)순으로 바뀝니다.

- 마찬가지로 La가 00, 01, 10인 동안에는 Bb가 11이어야 합니다.

- 다음과 같은 순서로 진행됩니다.



- 다음과 같은 형태로 표현됩니다.

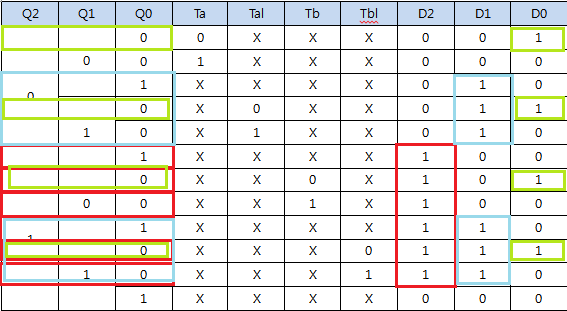


**<State Trasition Table>**

|  |  |
| --- | --- |
| State | Code |
| S0 | 000 |
| S1 | 001 |
| S2 | 010 |
| S3 | 011 |
| S4 | 100 |
| S5 | 101 |
| S6 | 110 |
| S7 | 111 |

- 입출력에 대한 Transition Table은 다음과 같습니다.

**<State Transition Table>**



- 위의 진리표를 통해서 D를 계산하면 다음과 같습니다.

D2 = (~Q2 \* Q1 \* Q0) + (Q2 \* ~Q1) + (Q2 \* Q1 \* ~Q0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 1101 | 1111 | 1110 | 1010 | 1011 | 1001 | 1000 |
| 000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 011 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 010 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 110 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 111 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 101 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 100 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

D1 = (~Q2 \* ~Q1 \* Q0) + (Q1 \* ~Q0) + (Q2 \* ~Q1 \* Q0)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 1101 | 1111 | 1110 | 1010 | 1011 | 1001 | 1000 |
| 000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 001 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 010 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 110 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 111 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 101 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 100 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

D0 = (~Q2 \* ~Q1 \* ~Q0 \* ~Ta) + (~Q2 \* Q1 \* ~Q0 \* ~Tal) + (Q2 \* ~Q1 \* ~Q0 \* ~Tb) + (Q2 \* Q1 \* ~Q0 \* ~Tbl)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 1101 | 1111 | 1110 | 1010 | 1011 | 1001 | 1000 |
| 000 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 001 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 011 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 010 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 110 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 111 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 101 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 100 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

**<Output Table>**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Q2 | Q1 | Q0 | La1 | La0 | Lb1 | Lb0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 |

-위의 진리표를 바탕으로 boolean 대수를 산출하면 다음과 같습니다.

La1 = (~Q2\*Q1\*~Q0) + (Q2\*~Q1\*~Q0) + (Q2\*~Q1\*Q0) + (Q2\*Q1\*~Q0) + (Q2\*Q1\*Q0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

La0 = (~Q2\*~Q1\*Q0) + (~Q2\*Q1\*Q0) + (Q2\*~Q1\*~Q0) + (Q2\*~Q1\*Q0) + (Q2\*Q1\*~Q0) + (Q2\*Q1\*Q0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Lb1 = (~Q2\*~Q1\*~Q0) + (~Q2\*~Q1\*Q0) + (~Q2\*Q1\*~Q0) + (~Q2\*Q1\*Q.0) + (Q2\*Q1\*~Q0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |

Lb0 = (~Q2\*~Q1\*~Q0) + (~Q2\*~Q1\*Q0) + (~Q2\*Q1\*~Q0) + (~Q2\*Q1\*Q0) + (Q2\*~Q1\*Q0) + (Q2\*Q1\*Q0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 00 | 01 | 11 | 10 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 |

- 사용된 모듈의 구성은 다음과 같습니다.

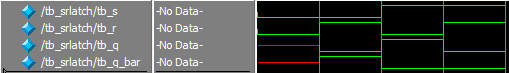
|  |  |  |
| --- | --- | --- |
| 구분 | 이름 | 설명 |
| Top module | tl\_cntl\_left | Traffic light controller의 top module |
| Sub module | ns\_logic | Traffic light controller의 next state를 결정하는 combinational logic |
| \_refister3\_r | 3bit resettable register with active low asynchronous reset module(내부에 dff\_r\_async를 instance) - 현재의 state를 저장 |
| \_dff\_r\_async | Resettable d flip flop with active low  asynchronous reset |
| o\_logic | 현재 state의 값에 기반하여 output 값을 결정하는 combinational logic |

-사용된 인자들의 구성은 다음과 같습니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 구분 | 이름 | 비트 수 (bit) | 설명 |
| Input | clk | 1 | Clock |
| Reset\_n | 1 | Active low에 동작하는 reset 신호로 state를 초기화함 |
| Ta | 1 | Traffic sensor A |
| Tal | 1 | Traffic sensor AL |
| Tb | 1 | Traffic sensor B |
| Tbl | 1 | Traffic sensor BL |
| Output | La | 2 | 신호등 값 출력 A |
| Lb | 2 | 신호등 값 출력 B |

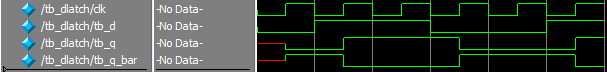
1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

1) SR latch



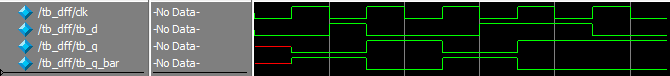
- 위와 같은 결과를 확인할 수 있었으며 Truth table과 동일한 결과를 보입니다.

2) D latch



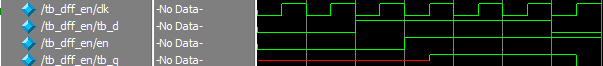
- 위와 같이 clock이 0일 때 이전의 값을 유지하는 것을 확인 할 수 있습니다.

3) D flip flop



- 위와 같이 clock이 0이면 이전의 값을 유지하고 1일 때 새로운 값을 받는 것을 확인할 수 있습니다.

4) enabled D flip flop



-위의 결과를 확인하면 en이 0일 때는 이전의 상태를 계속 유지하다가 1이 되고 clk가 1이 되면서 tb\_d값이 tb\_q로 나오는 것을 볼 수 있습니다. 이후 tb\_d가 0이 되면 그 다음 clk이 1이 될 때 tb\_q에 0이 나오는 것을 확인할 수 있었습니다.

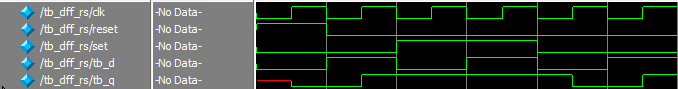
5) resettable D flip flop



- 위의 결과를 확인하면 reset이 0일 때는 정상적으로 입출력이 일어나지만 reset이 1이

되면서 결과가 0이 되는 것이 확인할 수 있습니다.

6) set/resettable D flip flop



- 위의 결과를 확인하면 reset이 0이 되면서 정상적으로 입출력이 일어나고 set이 1이 되면서 기존의 값을 계속 유지하다가 0이 되면서 다시 입력에 따라 출력이 바뀌는 것을

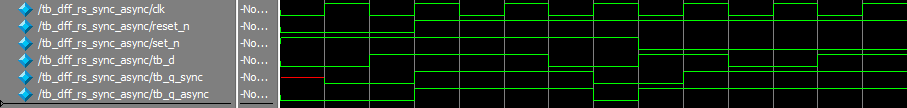
확인할 수 있었습니다.

7) 32bits register



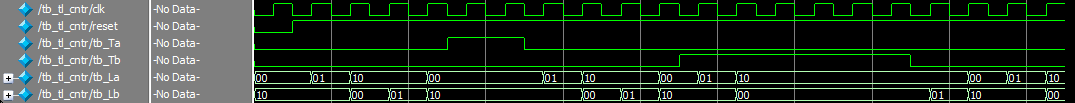
- 위를 확인하면 clk이 rising edge일 때 받아들인 값을 출력해주고 있음을 확인할 수 있습니다.

8) Asynchronous/Synchronous Set/Resettable d flip flop



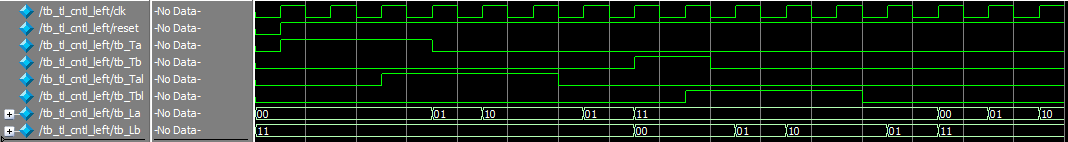
- 위의 결과를 보면 reset\_n이 1이 되면서 값들이 들어가는데 sync는 클럭의 rising edge에서만 결과값이 나오지만, async는 클럭의 rising edge와 reset\_n, set\_n의 falling edge에서도 결과 값이 나오게 된다. (40ns부분을 보면 확인가능) 이후 set\_n이 0이 되면서 값을 유지하는 모습을 확인할 수 있습니다.

9) Traffic light controller without left turn



- 위의 결과를 보면 Ta가 1이 되면 그만큼 00신호가 길어지며 그 다음 클럭의 rising edge에서 신호가 바뀌는 것을 확인할 수 있습니다. 그 시간만큼 Tb는 10신호를 이어가는 것 또한 확인할 수 있으며 그 반대의 경우도 같은 결과를 확인할 수 있습니다.

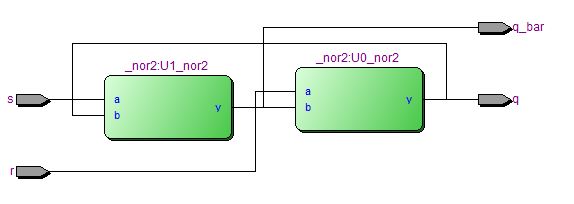
10) Traffic light controller with left turn



- 직진 신호에 대한 결과는 Traffic light controller without left turn과 동일하고 left turn이 있는 경우에 해당 신호가 끝나고 다음 rising edge까지 신호가 연속되는 것을 확인할 수 있습니다.

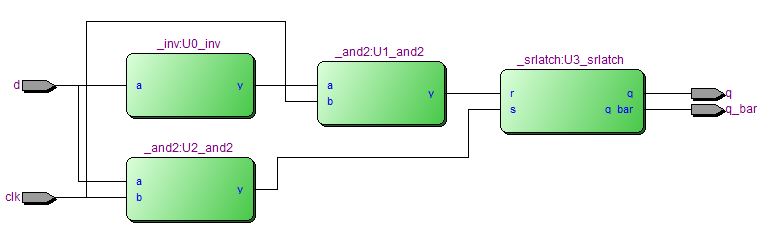
B. 합성(synthesis) 결과

1) SR latch



- nor게이트 2개를 이용해서 q, ~q를 도출합니다.

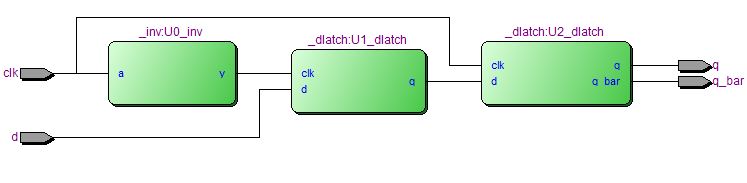
2) D latch



-클럭과 d의 and값, ~d와 클럭의 and 값이 and 게이트를 통해 들어가서 reset값이 되고

클럭과 d의 and 값이 set값이 됩니다.

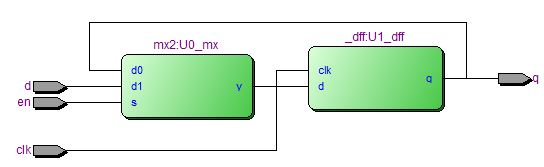
3) D flip flop



- ~clk와 d를 d latch에 넣고 그 결과값과 clk를 또 하나의 d latch에 넣어서 Q와 ~Q를

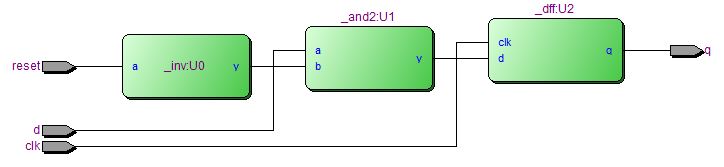
산출합니다.

4) enabled D flip flop



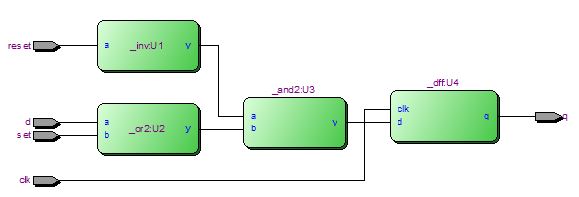
- mux2 module을 이용해서 어떤 값을 DFF에 input으로 할 것인지를 결정합니다.

5) resettable D flip flop



- ~reset(reset\_n)을 이용해서 input값을 초기화 시켜줄 수 있습니다.

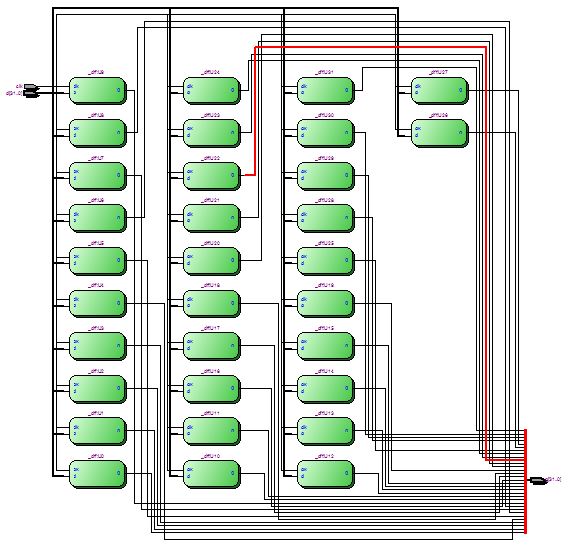
6) Set/Resettable D flip flop



- reset값을 통해서 입력을 초기화 시킬 수 있고, set을 통해서 입력된 값의 출력을

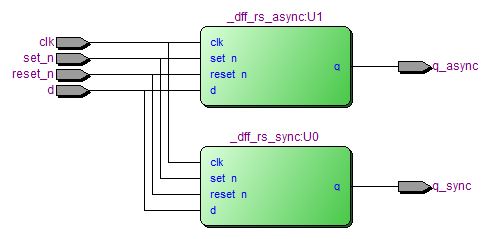
유지할 수 있습니다.

7) 32bits register



- 1bit register 32개를 instance함으로써 32bit 기억소자를 구성하였습니다.

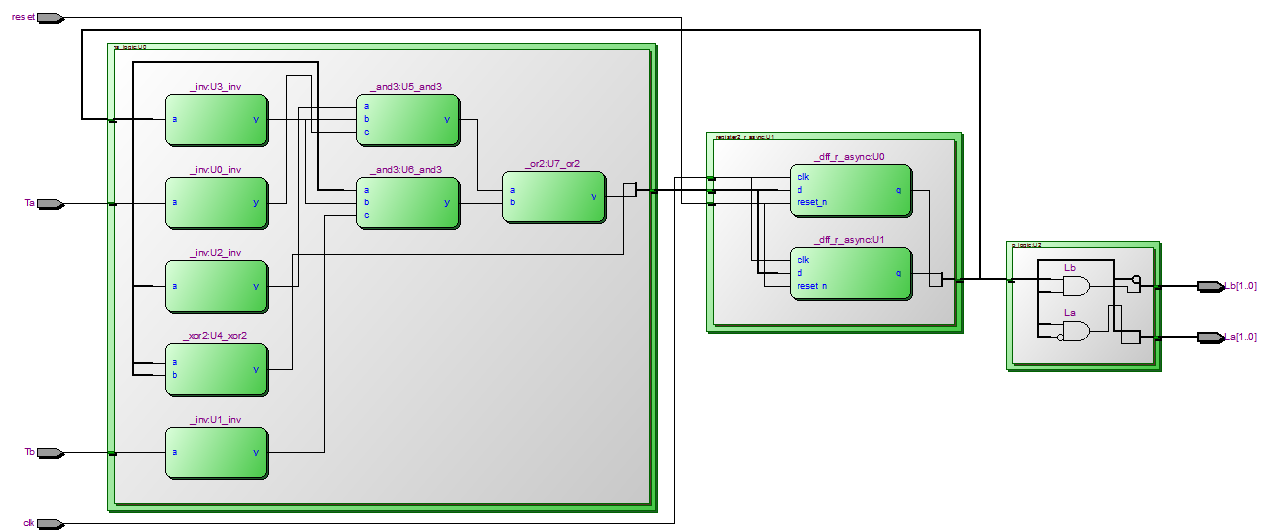
8) async/sync set/resettable D flip flop



- 위의 형태와 같이 두 개의 모듈을 이용하여 async값과 sync값을 확인할 수 있습니다.

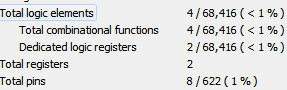
9) Traffic light controller without turn left

**<RTL Viewer>**



- 위와 같이 크게 3개의 모듈(ns\_logic, register2, o\_logic)으로 구성되있습니다.

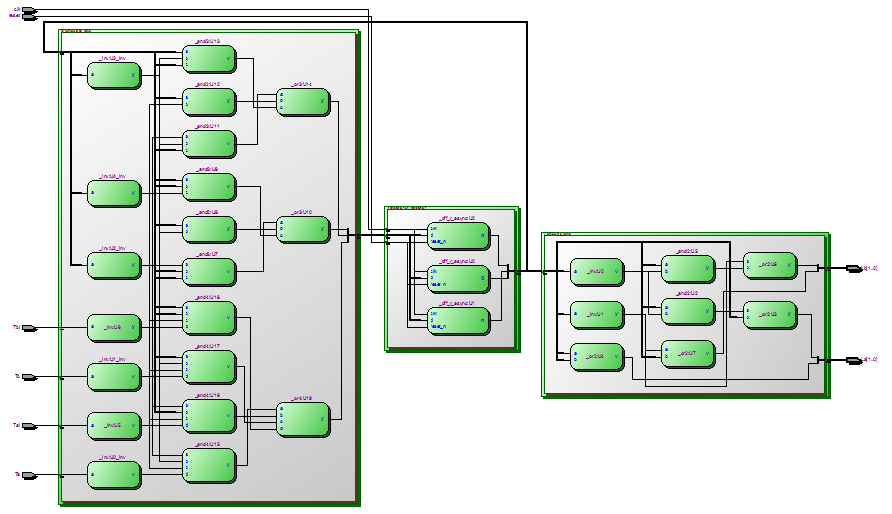
**<flow summary>**



**-** 4개의 logic이 사용되었고, dff\_r\_async를 register2에서 두번 instance했기 때문에 2개의 reg가 사용되었으며, 1bit input clk, Ta, Tb, reset, 2bit output La, Lb해서 총 8개의 pin이

사용되었습니다.

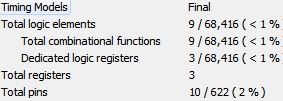
10) Traffic light controller with turn left



- 동일하게 크게 3개의 모듈로 나누어져 있으나 ns\_logic의 input이 2개 추가되었으며

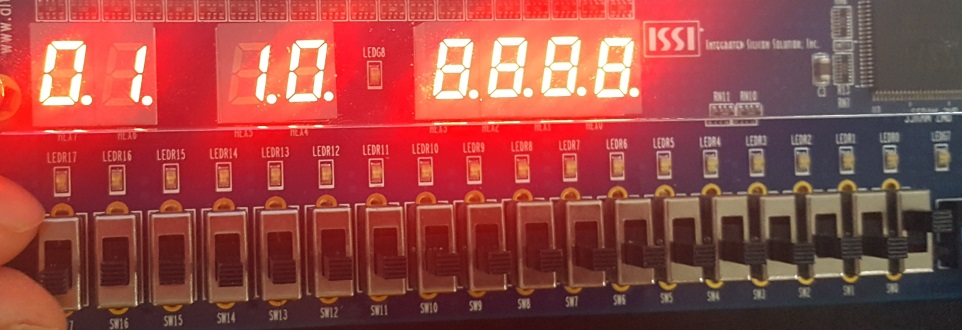
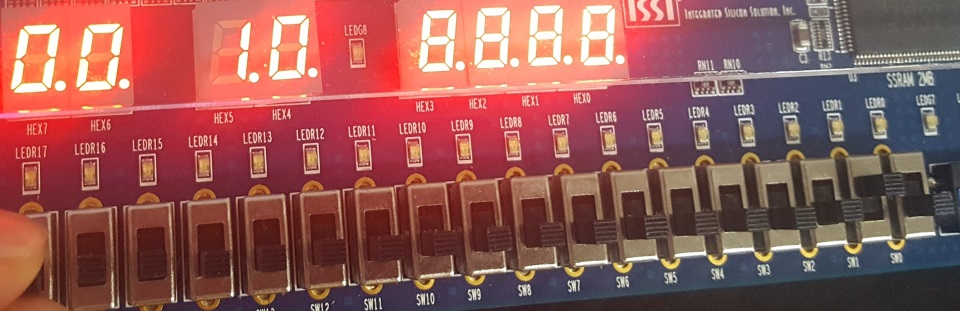
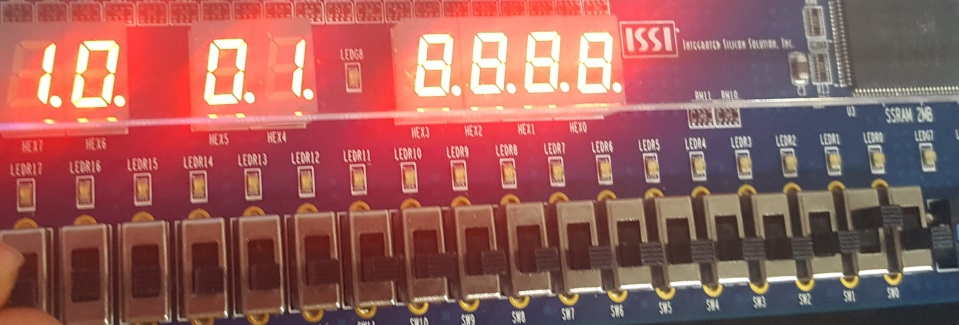
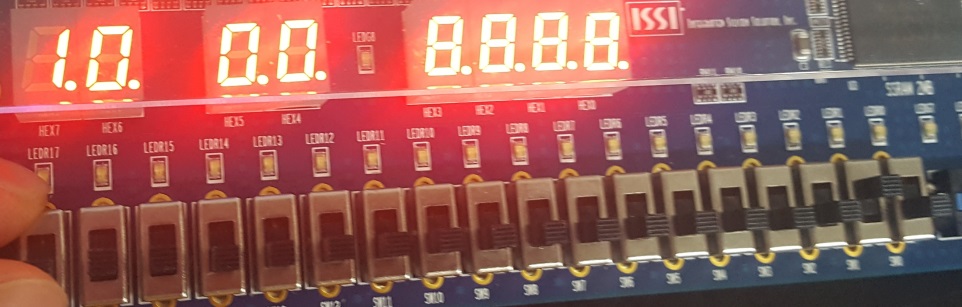
Register가 3bit를 기억하고 있음을 확인할 수 있었습니다.

**<Flow summary>**



- 총 9개의 logic이 사용되었으며, dff\_r\_async를 register3에서 세번 instance했기 때문에 3개의 reg가 사용되었고, 1bit input clk, reset, Ta, Tb, Tal, Tbl, 2bit output La, Lb 가 사용되어 총 10개의 pin이 사용되었습니다.

1. DE2-70 boarding (tl\_cntr)



1. B 차선에 빨간불, A 차선에 파란불

2. B 차선에 빨간불, A 차선에 노란불

3. B 차선에 파란불, A 차선에 빨간불

4. B 차선에 노란불, A 차선에 빨간불

5. B 차선에 빨간불, A 차선에 파란불

1. 고찰 및 결론

이번 실험을 진행하면서 register의 입.출력인 d0, d1,q1,q0를 d, ~d, q, ~q로 잘못보고

진행하다가 막혔었습니다. 솔직히 말해서 그때까지만 해도 이게 뭔지 어떻게 이어지는

건지 별로 이해가 안됐었는데, 몇 시간 동안이나 잡고 있었더니 그제서야 전체적인

그림이 그려지더군요. 역시 시간과의 싸움인 것 같습니다. 다시 정신차리고 2bit 입출력으로 수정하고서 진행하다 보니 이번에는 waveform에서 output 들이 모두 빨간줄로

뜨는 현상이 발견되었습니다. 이 또한 RTL viewer와 코드를 자세히 보니 오타와 함께

Wire를 잘못 연결하는 실수를 찾아내었습니다. 새삼 visual studio의 위대함을 깨달았습니다. 그 다음으로 어려웠던 부분은 case 문을 사용한 behavior design으로 tl\_cntr\_left를 구현할 때였습니다. 이 부분에서 13 이형민의 도움을 받았습니다.

1. 참고문헌

Wiki 백과사전

13 이형민의 도움을 받아 case를 이용한 tl\_cntr\_left를 구현하였습니다.