

과 목 명 : 디지털 시스템

담당교수 : 김종태 교수님

학 과 : 전자전기공학과

학 년 : 3학년

학 번 : 2018312121

이 름 : 장재성

과 제 명 : Assignment3 보고서

제 출 일 : 2022/05/19 (목)

1. **설계 목표**

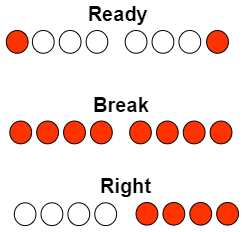
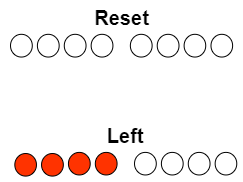
FSM(Finite State machine) 중에서 Moore State Machine을 이용하여 입력이 CLK, Reset, Break, Left, Right로, 출력이 8-bit의 LED로 구성된 Tail Light Controller(자동차 후미등 제어기)를 설계한다. 이 때, 최소 개의 state를 사용한다. 구현하는 과정에서 State diegram을 그린다. 구현 후에는 주어진 Test bench로 시뮬레이션으르 진행하여 검증한다.

1. **이론적 접근**
2. **“Tail Light Controller” 작동 방식**

Tail Light Controller의 동작은 크게 다음 4가지로 구성된다.

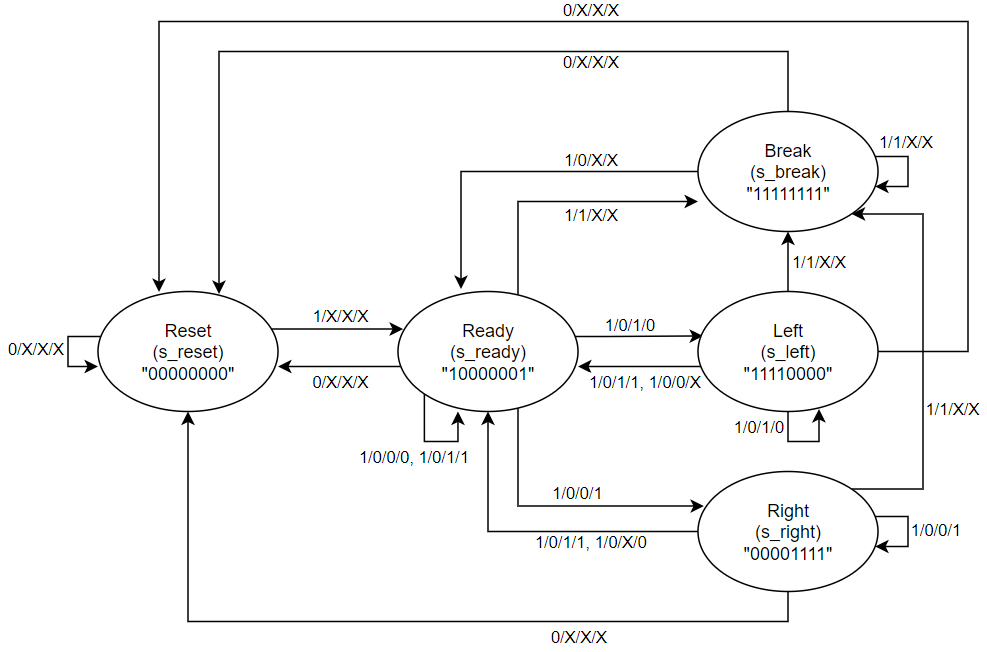
* Reset 동작을 제외한 모든 동작은 CLK의 Rising Edge에 맞춰 synchronous하게 동작한다.
* Reset은 0이 입력되는 활성화되는 Active low로, asynchronous하게 동작한다.
* Break, Left, Right는 1이 입력되면 활성화되는 Active High로, synchronous하게 동작한다.
* 입력은 한 번에 여러 개가 동시에 입력될 수 있다. (예를 들어, left와 right가 동시에 입력이 가능하다)

그리고 Tail Light는 현재 어떤 상태인지(동작) (Reset인지, Ready인지, Break인지, Left인지, Right인지) LED를 통해 나타낸다. 각 상태에 따른 LED 표현 방식은 다음과 같다.



1. **Tail Light Controller 무어 상태기계**

현재 상태에 의해서만 결과가 결정되는 Moore State machine의 State diagram에서 각 상태는 고유의 값을 지니며, State transition을 나타낼 때, State와 함께 output(Z)을 표시한다. Tail Light Controller의 동작을 고려하여 Moore State machine의 State diagram을 그리면 총 5개 State로 구성할 수 있다. Tail Light Controller의 Moore State machine State diagram은 다음과 같다.



각 상태에 대한 설명은 다음과 같다.

Reset(s\_reset) : Reset 상태는 어떤 상태에서든 RESET 입력 신호가 0인 경우 transition되는 “Initial State”에 해당한다. 즉, Reset이 일어날 때의 경우를 뜻한다. RESET 상태에 대한 동작은 다음과 같다.

* Reset은 CLK에 관계없이(asynchronous) 동작한다.
* Reset이 활성화되면 RESET 상태에 머무른다.
* Reset이 비활성화되면 다른 입력 값에 관계없이 Ready(s\_ready)상태가 된다. (synchronous)

Reset 상태는 위 동작을 따르며, Moore machine에서 값은 “00000000”이다. 이는 실제 자동차 후미등 전부가 꺼진 상태와 유사하다.

Ready(s\_ready) : Ready 상태는 Reset 상태에서 Reset 입력 신호가 1인 경우, Break 상태에서 Break 입력 신호가 0인 경우, Left 상태에서 Reset 신호를 제외한 입력 신호 중 Left 입력 신호만 1이 아닌 경우, Right 상태에서 Reset 신호를 제외한 입력 신호 중 Right 입력 신호만 1이 아닌 경우에 transition되는 상태이다. Ready 상태에서는 어떤 입력을 받느냐에 따라 “Reset”, “Break”, “Left”, “Right” 어떤 상태로든 transition이 일어날 수 있다. 즉, 어떤 상태 혹은 어떤 동작으로든 바뀔 수 있는 준비가 되어있는 경우를 뜻한다. Ready 상태에 대한 동작은 다음과 같다.

* Break가 입력되면 Reset을 제외한 다른 입력 값에 상관없이 Break 상태가 된다.
* Left만 입력되면 Left 상태가, Right만 입력되면 Right 상태가 된다.
* 다른 경우에서는 Ready 상태에 머무른다.

Ready 상태는 위 동작을 따르며, Moore machine에서 값은 “10000001”이다.

Break(s\_break) : Break 상태는 Ready 상태, Break 상태, Left 상태, Right 상태에서 Break 신호가 1인 경우에 transition되는 상태이다. Break 신호가 계속 1을 유지한다면, Left, Right 신호의 값에 상관없이 계속 Break 상태를 유지한다. 실제 자동차 후미등의 경우, 자동차의 브레이크를 밟았을 때 나타나는 경우이다. Break 상태에 대한 동작은 다음과 같다.

* Break가 입력되면 Reset을 제외한 다른 입력 값에 상관없이 break 상태에 머무른다.
* Break가 입력되지 않으면 Reset을 제외한 다른 입력 값에 상관없이 Ready 상태가 된다.

Break 상태는 위 동작을 따르며, Moore machine에서 값은 “11111111’이고, 이는 실제 자동차 후미등 전부가 켜진 상태와 유사하다.

Left(s\_left) : Left 상태는 Ready상태, Left 상태에서 Reset 신호를 제외하고 나머지 신호 중 Left 신호만 1인 경우에 transition 되는 상태이다. Left 신호가 계속 1을 유지한다면, 계속 Left 상태를 유지한다. 실제 자동차 후미등의 경우, 좌회전 신호를 넣었을 때 나타나는 경우이다. Left 상태에 대한 동작은 다음과 같다.

* Break가 입력되면 reset을 제외한 다른 입력 값에 상관없이 Break 상태로 간다.
* Left만 입력되면 Left 상태에 머무른다.
* 다른 경우에서는 Ready 상태로 간다.

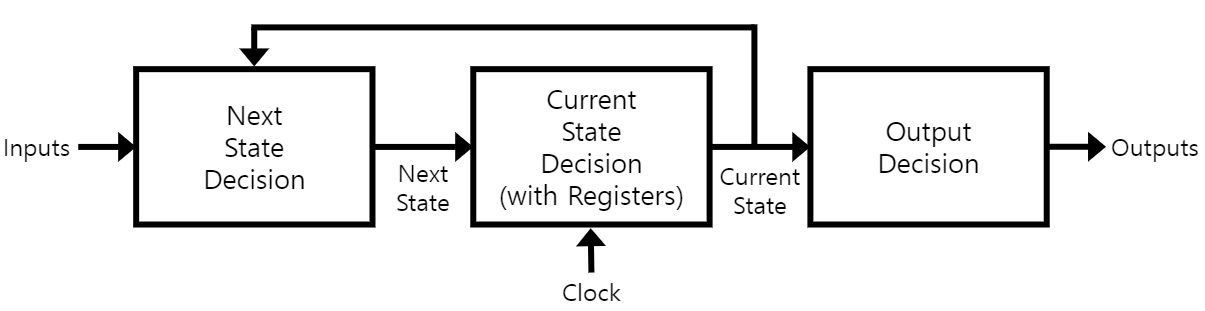
Left 상태는 위 동작을 따르며, Moore machine에서 값은 “11110000”이고, 이는 실제 자동차 후미등에서 왼쪽 부분만 켜진 상태와 유사하다.

Right(s\_right) : Right 상태는 Ready 상태, Right 상태에서 Reset 신호를 제외하고 나머지 신호 중 Right 신호만 1인 경우에 transition되는 상태이다. Right 신호가 계속 1을 유지한다면, 계속 Right 상태를 유지한다. 실제 자동차 후미등의 경우, 우회전 신호를 넣었을 때 나타나는 경우이다. Right 상태에 대한 동작은 다음과 같다.

* Break가 입력되면 Reset을 제외한 다른 입력 값에 상관없이 Break 상태로 간다.
* Right만 입력되면 Right 상태에 머무른다.
* 다른 경우에서는 Ready 상태로 간다.

Right 상태는 위 동작을 따르며, Moore machine에서 값은 “00001111”이고, 이는 실제 자동차 후미등에서 오른쪽 부분만 켜진 상태와 유사하다.

Moore machine은 ‘Next State Decision’, ‘Current State Decision’, ‘Output Deicision’ 총 3가지 부분으로 나뉜다. 여기서 ‘Next State Decision’과 ‘Output Decision’ 부분은 Combinational logic이고, ‘Current State Decision’ 부분은 CLK Event에 동작하는 Sequential logic이다. 전체적인 구성은 다음과 같다.



VHDL로 Tail Light Controller의 Moore machine 구현은 위와 같이 2개의 Combinational logic circuit과 1개의 Sequential logic circuit을 위에서 그린 Moore machine의 state diagram을 참고하여 구현함으로써 이루어진다. 이에 대해서는 추후 ‘VHDL 구현’ 부분에 자세히 설명한다.

1. **VHDL 구현(Source Code 설명)**
2. **Tail Light Controller 무어 상태기계 VHDL 구현**

우선, Input은 1-bit의 CLK, RESET, BREAK, LEFT, RIGHT을 선언하고, Output은 8-bit의 LED를 선언한다. 모든 input, output port는 ‘std\_logic’ type으로 선언한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 후, state에 대한 type 선언과 state에 대한 정보를 담을 내부 signal을 선언해야 한다. Tail Light Controller의 Moore machine state diagram에 따르면 state는 s\_reset(RESET / Initial state), s\_ready(READY), s\_break(BREAK), s\_left(LEFT), s\_right(RIGHT) 총 5개로 구성된다. 이 5개의 state에 대해 ‘states’라는 하나의 type을 선언하고, state에 대한 정보를 담을 변수를 앞서 선언한 states type으로 현재 상태를 나타내는 ‘state’와 다음 상태를 나타내는 ‘next\_state’ 두 개 선언한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

그 후, Moore machine을 구성하는3가지 부분을 구현한다.

첫 번째로, ‘Current State Decision’ 부분은 Reset 신호가 valid(‘0’)한 경우, State를 Initial State(s\_reset)로 transition하고, Reset 신호가 invalid(‘1’)한 경우, CLK에 따라 state가 다음 state로 transition하도록 구현한다. if문을 이용하여 RESET이 ‘0’인 경우(if (RESET = ‘0’)), state은 s\_reset를, RESET이 1이고 CLK이 rising edge인 경우(elsif (CLK = ‘1’ and CLK’event)), state는 next\_state를 할당하도록 설정한다. Process의 sensitivity list는 해당 프로세스에서 사용하는 입력이 CLK과 RESET이므로, ‘process(CLK, RESET)’로 작성한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

두 번째로, ‘Output Decision’ 부분은 Tail Light Controller Moore State diagram을 참고한다. Case 구문을 이용하여 모든 state에 대하여 각 state인 경우의 출력(LED)에 앞서 그렸던 Moore state diagram을 참고하여 알맞은 값을 할당하면 된다. 해당 부분은 combinational logic이고, Mealy machine은 현재 상태와 입력에 따라 Output이 결정되므로 always sensitivity에 CLK이 들어가지 않고, 다른 입력인 state만 쓰이면서 구현된다.

텍스트, 테이블이(가) 표시된 사진

자동 생성된 설명

세 번째로, ‘Next state’ 부분은 Tail Light Controller Moore State diagram을 참고한다. Case 구문을 이용하여 모든 state에 대하여 각 state인 경우 다양한 입력에 따른 다음 상태를 앞에서 그렸던 Tail Light Controller Moore state diagram을 참고하여 알맞은 값을 할당하면 된다. 각 state마다 RESET을 제외한 입력 BREAK, LEFT, RIGHT 신호가 0인지 1인지에 따라 next\_state의 값이 달라지므로, 각 state를 서술할 때마다 if 구문을 사용하여 BREAK, LEFT, RIGHT를 Tail Light Controller의 동작을 참고하여 0과 1의 경우로 나누어 next\_state에 값을 다르게 할당해야 한다. 해당 부분은 combinational logic이고, Moore machine은 현재 상태에 따라서 Output이 결정되므로 always sensitivity에 CLK이 들어가지 않고, 다른 입력인 state, BREAK, LEFT, RIGHT만 쓰이면서 구현된다.

텍스트이(가) 표시된 사진

자동 생성된 설명 텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

(각 State에 대한 설명에 대한 주석은 VHDL 코드에 쓰여있다)

1. **검증 계획 (Test bench code 설명)**

Tail Light Controller 검증은 앞서 구현한 entity(TAIL\_LIGHT)에 대한 component 선언 후 port out하고, 입력들(CLK, RESET, BREAK, LEFT, RIGHT)을 지연 시간을 활용하여 알맞은 시간에 알맞은 값으로 할당하여 그에 따른 결과 LED가 알맞게 출력되는지 확인함으로써 이루어진다. 이에 대한 Test bench code를 만들어서 검증을 해야 한다. Test Bench Code는 다음과 같다.

1. **Tail Light Controller 무어 상태기계 Test Bench Code**

Test Bench Code는 외부 I/O signal을 선언하지 않고, 모두 내부 signal을 선언하고 값을 설정함으로써 이루어진다. 즉, Test에 대한 entity를 선언할 때 I/O port부분을 작성하지 않는다. 그 후, architecture부분에 앞서 구현한 Tail Light Controller entity(TAIL\_LIGHT)를 대상으로 component 선언을 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

입력에 대한 값을 설정하고 출력에 대한 값을 저장할 내부 signal을 선언한다. 그 후, 앞서 선언한 Tail Light Controller Moore machine의 component인 TAIL\_LIGHT를 해당 component의 I/O port와 내부 signal을 알맞게 연결하여 Port map(Instantiation)한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

그 후, RESET, CLK, BREAK, LEFT, RIGHT(Input)에 대한 값들을 설정한다. 각각의 입력에 대한 설정은 각 입력마다 process 구문을 활용하고 해당 process문 안에서 ‘wait for’ 구문을 활용하여 원하는 시간에 원하는 값이 할당되도록 한다.

우선, RESET의 입력 값은 10ns일 때 ‘0’, 25ns일 때 ‘1’, 73ns일 때 ‘0’, 85ns일 때 ‘1’이 되도록 설정한다. 중간에 ‘wait for’ 구문을 통해 지연 시간을 설정함으로써 위의 test process를 구성할 수 있다. 이 부분에 대한 RESET의 Test process와 이에 대한 VHDL 코드만 다음과 같이 나타낸다.

소스로 주어진 RESET의 Test process

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(ns) | 0 | 10 | 25 | 73 | 85 |
| RESET | ‘X’ | ‘0’ | ‘1’ | ‘0’ | ‘1’ |

텍스트이(가) 표시된 사진

자동 생성된 설명

다음으로, CLK은 주기를 10ns로 설정한다. 이를 설정하기 위해 처음 5ns 동안에는 CLK 값이 ‘1’로 설정하고, 다음 5ns 동안에는 CLK값이 ‘0’으로 설정한다. 이 과정이 Simulation 동안 반복이 되어 주기가 10ns인 CLK이 만들어진다. 이에 대한 VHDL 코드는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

BREAK, LEFT, RIGHT(Input)에 대한 값 설정은 RESET의 값 설정과 동일한 방식으로 이루어지므로 자세한 설명은 RESET의 값 설정 부분을 참고한다. 이 부분에 대한 BREAK, LEFT, RIGHT의 Test process와 이에 대한 VHDL 코드만 다음과 같이 나타낸다.

BREAK의 Test process

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time(ns) | 0 | 10 | 25 | 35 | 105 | 115 |
| BREAK | ‘U’ | ‘0’ | ‘1’ | ‘0’ | ‘1’ | ‘0’ |

LEFT의 Test process

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time(ns) | 0 | 10 | 35 | 55 |
| X | ‘U’ | ‘0’ | ‘1’ | ‘0’ |

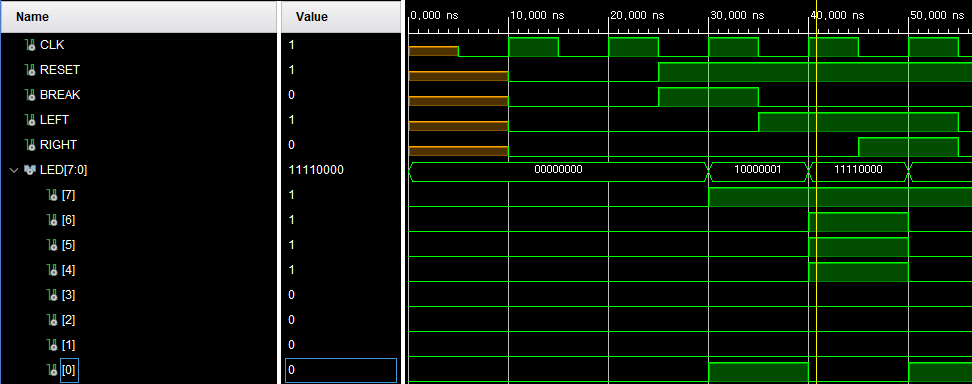
RIGHT의 Test process

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Time(ns) | 0 | 10 | 45 | 55 | 65 | 145 |
| X | ‘U’ | ‘0’ | ‘1’ | ‘0’ | ‘1’ | ‘0’ |

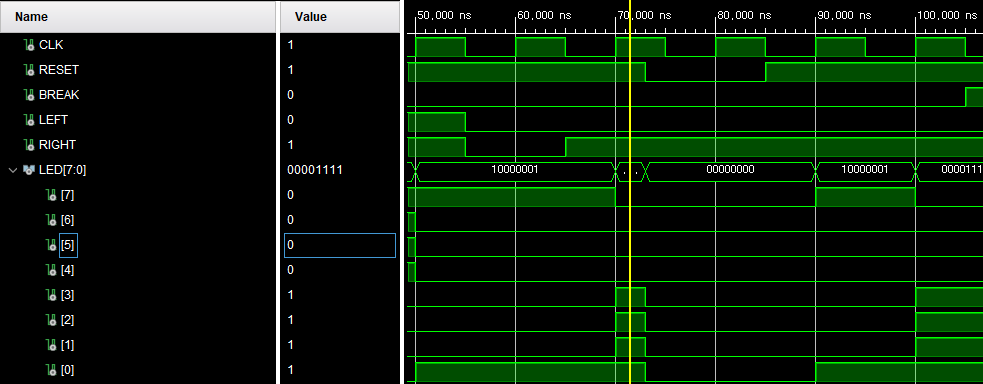
텍스트이(가) 표시된 사진

자동 생성된 설명

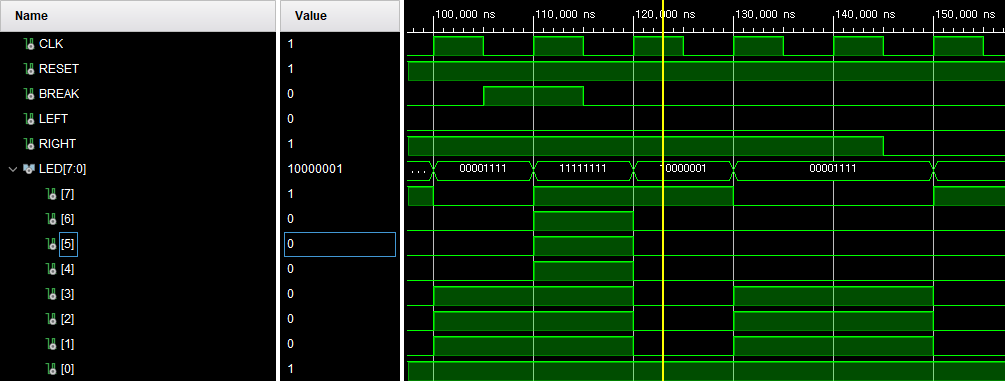
1. **시뮬레이션 결과 및 분석**
2. **Tail Light Controller 무어 상태기계 시뮬레이션 결과 및 분석**



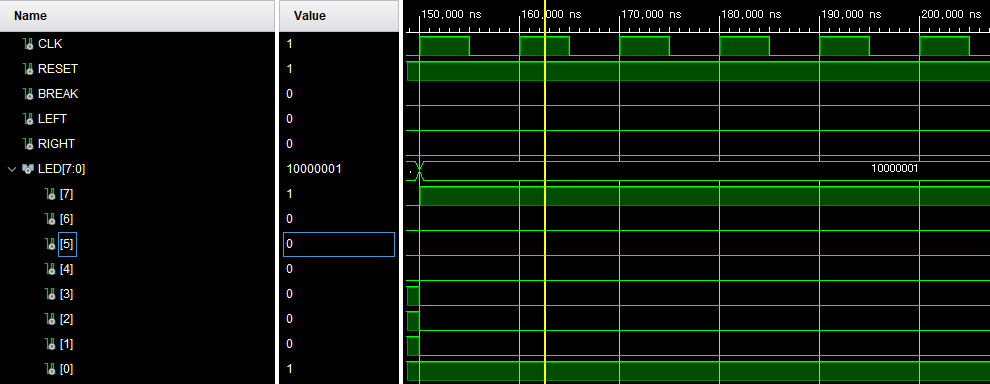
<Tail Light Controller Simulation 0ns ~ 50ns의 결과>



<Tail Light Controller Simulation 50ns ~ 100ns의 결과>



<Tail Light Controller Simulation 100ns ~ 150ns의 결과>



<Tail Light Controller Simulation 150ns ~ 200ns의 결과>

위의 Simulation waveform을 보면 CLK의 주기가 10ns인 것을 볼 수 있다. 그리고, 처음 0ns에는 모든 입력들에 대한 값이 설정되지 않아 ‘U’로 설정됨을 볼 수 있다. 10ns부터 25ns까지 RESET 값이 ‘0’이므로, 이 때, Tail Light Controller는 RESET 동작을 하며, 상태는 초기 상태인 s\_reset(RESET) 상태가 설정되며 출력도 “00000000”으로 설정된다. 25ns 이후 RESET 값이 ‘1’이 되면서 RESET 동작은 일어나지 않는 대신 CLK 신호가 trigger(rising edge)되는 시점마다 상태는 입력 BREAK, LEFT, RIGHT값에 따라 알맞은 상태로 transition이 일어나고, 현재 상태에 따라 출력 LED의 8 bit 출력 값이 결정된다. 25ns 이후 RESET 신호가 다시 ‘0’이 되는 75ns까지 입력 BREAK, LEFT, RIGHT, 상태(S), 그리고 출력(LED)에 대한 표를 그려보면 다음과 같다. 중간에 상태와 값이 바뀌는 부분은 CLK이 rising edge trigger되었을 때 나타나는 변화이다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time(ns) | 25~35 | 35~45 | 45~55 | 55~65 | 65~75 |
| BREAK | ‘1’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ |
| LEFT | ‘0’ | ‘1’ | ‘1’ | ‘0’ | ‘0’ |
| RIGHT | ‘0’ | ‘0’ | ‘1’ | ‘0’ | ‘1’ |
| State | s\_reset  ->  s\_ready | s\_ready  ->  s\_left | s\_left  ->  s\_ready | s\_ready  ->  s\_ready | s\_ready  ->  s\_right |
| LED | “00000000”  ->  “10000001”“ | “10000001”  ->  “11110000” | “11110000”  ->  “10000001” | “10000001”  ->  “10000001” | “10000001”  ->  “00001111” |

그 후, 75ns 부터 85ns까지 RESET신호가 ‘0’이기 때문에 Tail Light Controller는 RESET 동작이 일어나 초기 상태(s\_reset)로 바뀌며, 출력 LED도 “00000000”으로 설정된다. RESET 신호가 다시 ‘1’이 되는 85ns부터 200ns까지 입력 BREAK, LEFT, RIGHT, State, 그리고 출력(LED)에 대한 표를 그려보면 다음과 같다.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Time(ns) | 85~95 | 95~105 | 105~115 | 115~125 | 125~140 | 140~145 | 145~150 | 150~200 |
| BREAK | ‘0’ | ‘0’ | ‘1’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ |
| LEFT | ‘0’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ | ‘0’ |
| RIGHT | ‘1’ | ‘1’ | ‘1’ | ‘1’ | ‘1’ | ‘1’ | ‘0’ | ‘0’ |
| State | s\_reset  ->  s\_ready | s\_ready  ->  s\_right | s\_right  ->  s\_break | s\_break  ->  s\_ready | s\_ready  ->  s\_right | s\_right | s\_right  ->  s\_ready | s\_ready |
| LED | “00000000”  ->  “10000001” | “10000001”  ->  “00001111” | “00001111”  ->  “11111111” | “11111111”  ->  “10000001” | “10000001”  ->  “00001111” | “00001111” | “00001111”  ->  “10000001” | “10000001” |

위에서 분석한 결과들을 보면, Tail Light Controller의 RESET 동작과 CLK과 입력 BREAK, LEFT, RIGHT에 따른 상태변화가 State diagram에 맞게 작동하여 결과적으로 Tail Light Controller의 동작을 제대로 수행함을 알 수 있다. 자세히는 다음과 같다.

* s\_reset으로 transition인 경우(RESET 동작) 검증 : 10ns~25ns / 75ns ~ 85ns
* s\_ready로 transition인 경우(READY 동작) 검증 : 25ns~35ns / 45ns ~ 55ns / 55ns~65ns / 85ns ~ 95ns / 115ns~125ns / 145ns~150ns
* s\_break로 transition인 경우(BREAK 동작) 검증 : 105ns~115ns
* s\_left로 transition인 경우(LEFT 동작) 검증 : 35ns~45ns
* s\_right로 transition인 경우(RIGHT 동작) 검증 : 65ns~75ns / 95ns~105ns / 125ns~140ns

1. **고찰**

이번 과제에서는 Tail Light Controller의 작동 방식을 이해하고 이를 바탕으로 Moore machine을 Tail Light Controller를 대상으로 VHDL로 구현하고, 구현한 Moore machine을 호출(Port Map)하여 Tail Light Controller를 구현하였다. 구현 과정 중에서는 Tail Light Controller의 Moore machine에 대한 State diagram을 그려서 작동 방식에 대해 먼저 설계하였다. 특히 이번 설계 과제는 실생활에 적용되는 동작에 대한 구현으로, 전자 부품에 들어있는 디지털 회로를 설계하기 위해서 이런 과정을 거쳐 설계되고 구현된다는 것을 느낄 수 있었다. 그리고, FSM의 개념을 실생활과 연관지어 구성함으로써 FSM의 개념은 단순히 하드웨어적, 소프트웨어적 개념에 국한되는 것이 아니라 굉장히 다양한 분야에 쓰일 수 있음을 알 수 있었다.

이번 과제를 통해서 추후 실생활과 관련된 동작에 대해 디지털 회로를 설계할 때 동작에 대한 상세한 이해, 동작을 구현하기 전에 해당 동작에 대한 FSM과 같은 논리적 개념을 이용하여 이론적으로 나타내는 부분의 중요성을 깨달았고, 추후 어떤 하드웨어를 구현할 때마다 단순히 코드 그 자체에만 집중하기 보다 코드를 짜기 전 이론적 부분에 집중해야겠다는 생각이 들었다.

이 생각은 하드웨어나 소프트웨어를 설계할 때 많은 도움이 될 것이다.