7주차 결과보고서

전공: 수학과 학년: 4학년 학번: 20171273 이름: 심현우

1. **Even Parity bit generator 및 checker의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 K-map 포함)**

**-Generator**

**텍스트이(가) 표시된 사진

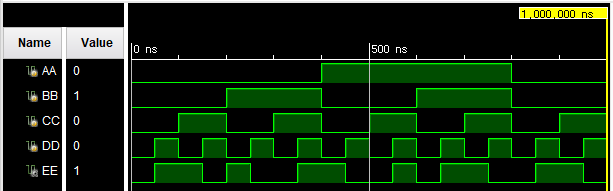
자동 생성된 설명**

PBG.v

**텍스트이(가) 표시된 사진

자동 생성된 설명**

PBG\_tb.v



Simulation

Even parity bit generator를 보자. Parity bit generator는 parity bit를 이용하여 데이터를 송수신할 때 오류를 확인하기 위해 여분의 bit를 생성하는 기계이다. Even parity bit generator는 수신하는 쪽에서 짝수개의 1을 수신할 수 있도록 송신하는 쪽에서 홀수개의 데이터를 보낼 시 parity bit를 1로 하여 보낸다. 따라서 input으로 4자리 bit가 들어가고 output으로느 parity bit가 있다. Parity bit generator는 output을 parity bit로 한다. 따라서 이를 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **Out E** |
| **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **1** | **1** |
| **0** | **0** | **1** | **0** | **1** |
| **0** | **0** | **1** | **1** | **0** |
| **0** | **1** | **0** | **0** | **1** |
| **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **1** |
| **1** | **0** | **0** | **0** | **1** |
| **1** | **0** | **0** | **1** | **0** |
| **1** | **0** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** | **0** |

진리표를 이용하여 카르노 맵을 작성해보자

-E

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **0** | **1** |
| **01** | **1** | **0** | **1** | **0** |
| **11** | **0** | **1** | **0** | **1** |
| **10** | **1** | **0** | **1** | **0** |

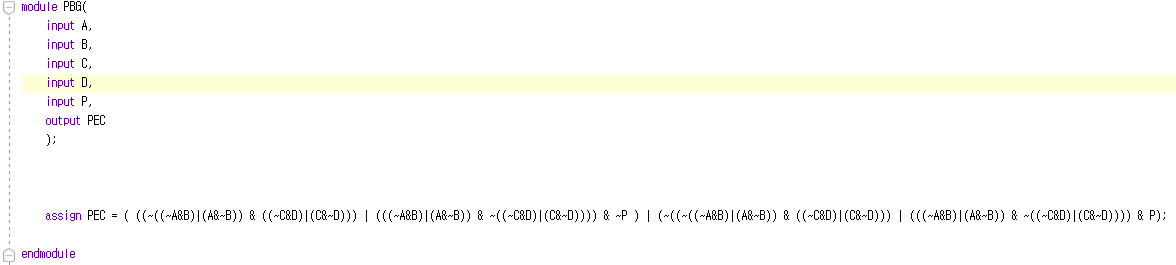
위의 카르노 맵을 왼쪽 밑 오른쪽 위를 묶고 나머지 하나씩 각각 하여 boolean 식을 작성하면 밑에와 같다.

E=

(~((~A&B)|(A&~B)) & ((~C&D)|(C&~D))) | (((~A&B)|(A&~B)) & ~((~C&D)|(C&~D)))

위의 Boolean 식을 verilog에 작성하여 simulation결과를 truth table과 비교해보자. 맨 위 사진과 simulation결과를 보면 카르노 맵을 그리기 위한 truth table과 일치하는 것을 확인할 수 있다.

**-Checker**

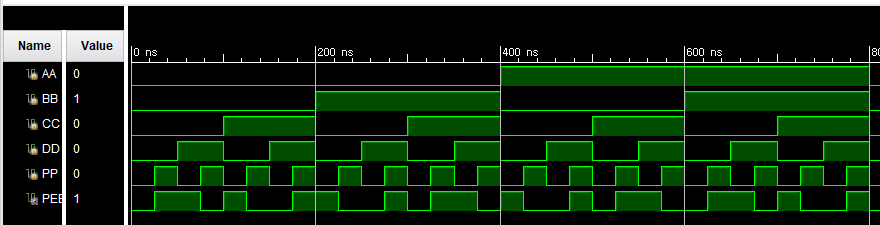


PBC.v

텍스트이(가) 표시된 사진

자동 생성된 설명

PBC\_tb.v



Simulation

Even parity bit checker를 보자. Parity bit checker는 parity bit를 이용하여 데이터를 수신할 때 오류를 확인하기 위해 만들어진 여분의 bit를 판독하는 기계이다. Even parity bit checker는 송신하는 쪽에서 보낸 parity bit와 데이터를 이용하여 짝수개가 맞는지를 확인하여 오류가 없으면 0 있으면 1을 출력한다. 따라서 input으로 4자리 bit와 parity bit 하나가 들어가고 output으로느 check bit가 있다. 따라서 이를 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **In P** | **Out PEC** |
| **0** | **0** | **0** | **0** | **0** | **0** |
| **0** | **0** | **0** | **0** | **1** | **1** |
| **0** | **0** | **0** | **1** | **0** | **1** |
| **0** | **0** | **0** | **1** | **1** | **0** |
| **0** | **0** | **1** | **0** | **0** | **1** |
| **0** | **0** | **1** | **0** | **1** | **0** |
| **0** | **0** | **1** | **1** | **0** | **0** |
| **0** | **0** | **1** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **1** | **0** | **0** | **1** | **0** |
| **0** | **1** | **0** | **1** | **0** | **0** |
| **0** | **1** | **0** | **1** | **1** | **1** |
| **0** | **1** | **1** | **0** | **0** | **0** |
| **0** | **1** | **1** | **0** | **1** | **1** |
| **0** | **1** | **1** | **1** | **0** | **1** |
| **0** | **1** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **0** | **1** |
| **1** | **0** | **0** | **0** | **1** | **0** |
| **1** | **0** | **0** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** | **1** |
| **1** | **0** | **1** | **0** | **0** | **0** |
| **1** | **0** | **1** | **0** | **1** | **1** |
| **1** | **0** | **1** | **1** | **0** | **1** |
| **1** | **0** | **1** | **1** | **1** | **0** |
| **1** | **1** | **0** | **0** | **0** | **0** |
| **1** | **1** | **0** | **0** | **1** | **1** |
| **1** | **1** | **0** | **1** | **0** | **1** |
| **1** | **1** | **0** | **1** | **1** | **0** |
| **1** | **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **1** | **0** | **1** | **0** |
| **1** | **1** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **1** | **1** | **1** |

위 진리표를 통해 5변수 카르노맵을 작성하면 다음과 같다.

-P=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **0** | **1** |
| **01** | **1** | **0** | **1** | **0** |
| **11** | **0** | **1** | **0** | **1** |
| **10** | **1** | **0** | **1** | **0** |

-P=1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | **00** | **01** | **11** | **10** |
| **00** | **1** | **0** | **1** | **0** |
| **01** | **0** | **1** | **0** | **1** |
| **11** | **1** | **0** | **1** | **0** |
| **10** | **0** | **1** | **0** | **1** |

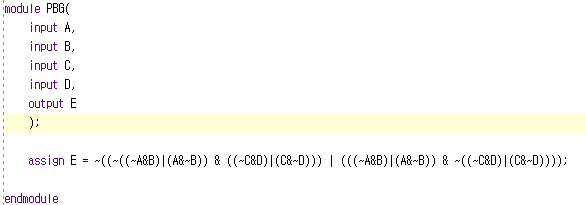
이 카르노 맵은 각 A, B, C, D를 XOR게이트로 묶은 것이다. 이를 Boolean 식으로 작성하면 다음과 같다.

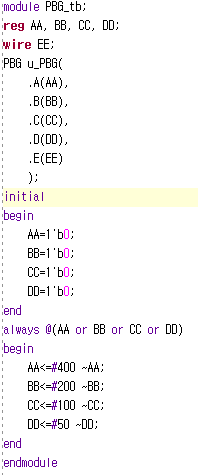
PEC = ( ((~((~A&B)|(A&~B)) & ((~C&D)|(C&~D))) | (((~A&B)|(A&~B)) & ~((~C&D)|(C&~D)))) & ~P ) | (~((~((~A&B)|(A&~B)) & ((~C&D)|(C&~D))) | (((~A&B)|(A&~B)) & ~((~C&D)|(C&~D)))) & P)

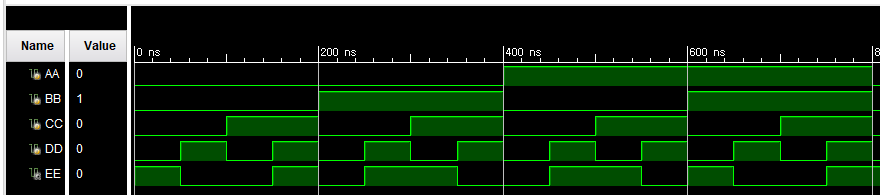
위의 Boolean 식을 verilog에 작성하여 simulation결과를 truth table과 비교해보자. 맨 위 사진과 simulation결과를 보면 카르노 맵을 그리기 위한 truth table과 일치하는 것을 확인할 수 있다.

1. **Odd Parity bit generator 및 checker의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 K-map 포함)**

**-Generator**







Odd parity bit generator를 보자. Parity bit generator는 parity bit를 이용하여 데이터를 송수신할 때 오류를 확인하기 위해 여분의 bit를 생성하는 기계이다. Odd parity bit generator는 수신하는 쪽에서 홀수개의 1을 수신할 수 있도록 송신하는 쪽에서 짝수개의 데이터를 보낼 시 parity bit를 1로 하여 보낸다. 따라서 input으로 4자리 bit가 들어가고 output으로느 parity bit가 있다. Parity bit generator는 output을 parity bit로 한다. 따라서 이를 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **Out E** |
| **0** | **0** | **0** | **0** | **1** |
| **0** | **0** | **0** | **1** | **0** |
| **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **1** | **1** | **1** |
| **0** | **1** | **0** | **0** | **0** |
| **0** | **1** | **0** | **1** | **1** |
| **0** | **1** | **1** | **0** | **1** |
| **0** | **1** | **1** | **1** | **0** |
| **1** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** |
| **1** | **0** | **1** | **0** | **1** |
| **1** | **0** | **1** | **1** | **0** |
| **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **0** | **1** | **0** |
| **1** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **1** | **1** |

진리표를 이용하여 카르노 맵을 작성해보자

-E

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | **00** | **01** | **11** | **10** |
| **00** | **1** | **0** | **1** | **0** |
| **01** | **0** | **1** | **0** | **1** |
| **11** | **1** | **0** | **1** | **0** |
| **10** | **0** | **1** | **0** | **1** |

위의 카르노 맵을 왼쪽 밑 오른쪽 위를 묶고 나머지 하나씩 각각 하여 boolean 식을 작성하면 밑에와 같다.

E=

~((~((~A&B)|(A&~B)) & ((~C&D)|(C&~D))) | (((~A&B)|(A&~B)) & ~((~C&D)|(C&~D))))

위의 Boolean 식을 verilog에 작성하여 simulation결과를 truth table과 비교해보자. 맨 위 사진과 simulation결과를 보면 카르노 맵을 그리기 위한 truth table과 일치하는 것을 확인할 수 있다.

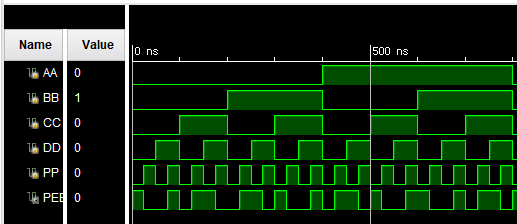
**-Checker**

**텍스트이(가) 표시된 사진

자동 생성된 설명**

**텍스트이(가) 표시된 사진

자동 생성된 설명**



Odd parity bit checker를 보자. Parity bit checker는 parity bit를 이용하여 데이터를 수신할 때 오류를 확인하기 위해 만들어진 여분의 bit를 판독하는 기계이다. Odd parity bit checker는 송신하는 쪽에서 보낸 parity bit와 데이터를 이용하여 홀수개가 맞는지를 확인하여 오류가 없으면 0 있으면 1을 출력한다. 따라서 input으로 4자리 bit와 parity bit 하나가 들어가고 output으로느 check bit가 있다. 따라서 이를 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **In P** | **Out PEC** |
| **0** | **0** | **0** | **0** | **0** | **1** |
| **0** | **0** | **0** | **0** | **1** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** |
| **0** | **0** | **0** | **1** | **1** | **1** |
| **0** | **0** | **1** | **0** | **0** | **0** |
| **0** | **0** | **1** | **0** | **1** | **1** |
| **0** | **0** | **1** | **1** | **0** | **1** |
| **0** | **0** | **1** | **1** | **1** | **0** |
| **0** | **1** | **0** | **0** | **0** | **0** |
| **0** | **1** | **0** | **0** | **1** | **1** |
| **0** | **1** | **0** | **1** | **0** | **1** |
| **0** | **1** | **0** | **1** | **1** | **0** |
| **0** | **1** | **1** | **0** | **0** | **1** |
| **0** | **1** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **1** | **0** | **0** |
| **0** | **1** | **1** | **1** | **1** | **1** |
| **1** | **0** | **0** | **0** | **0** | **0** |
| **1** | **0** | **0** | **0** | **1** | **1** |
| **1** | **0** | **0** | **1** | **0** | **1** |
| **1** | **0** | **0** | **1** | **1** | **0** |
| **1** | **0** | **1** | **0** | **0** | **1** |
| **1** | **0** | **1** | **0** | **1** | **0** |
| **1** | **0** | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **1** | **1** | **1** |
| **1** | **1** | **0** | **0** | **0** | **1** |
| **1** | **1** | **0** | **0** | **1** | **0** |
| **1** | **1** | **0** | **1** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** | **1** |
| **1** | **1** | **1** | **0** | **0** | **0** |
| **1** | **1** | **1** | **0** | **1** | **1** |
| **1** | **1** | **1** | **1** | **0** | **1** |
| **1** | **1** | **1** | **1** | **1** | **0** |

위 진리표를 통해 5변수 카르노맵을 작성하면 다음과 같다.

-P=0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | **00** | **01** | **11** | **10** |
| **00** | **1** | **0** | **1** | **0** |
| **01** | **0** | **1** | **0** | **1** |
| **11** | **1** | **0** | **1** | **0** |
| **10** | **0** | **1** | **0** | **1** |

-P=1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| AB  CD | **00** | **01** | **11** | **10** |
| **00** | **0** | **1** | **0** | **1** |
| **01** | **1** | **0** | **1** | **0** |
| **11** | **0** | **1** | **0** | **1** |
| **10** | **1** | **0** | **1** | **0** |

이 카르노 맵은 각 A, B, C, D를 XOR게이트로 묶은 것이다. 이를 Boolean 식으로 작성하면 다음과 같다.

PEC = ~(( ((~((~A&B)|(A&~B)) & ((~C&D)|(C&~D))) | (((~A&B)|(A&~B)) & ~((~C&D)|(C&~D)))) & ~P ) | (~((~((~A&B)|(A&~B)) & ((~C&D)|(C&~D))) | (((~A&B)|(A&~B)) & ~((~C&D)|(C&~D)))) & P))

위의 Boolean 식을 verilog에 작성하여 simulation결과를 truth table과 비교해보자. 맨 위 사진과 simulation결과를 보면 카르노 맵을 그리기 위한 truth table과 일치하는 것을 확인할 수 있다.

1. **2-bit binary comparator의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 K-map 포함)**

텍스트이(가) 표시된 사진

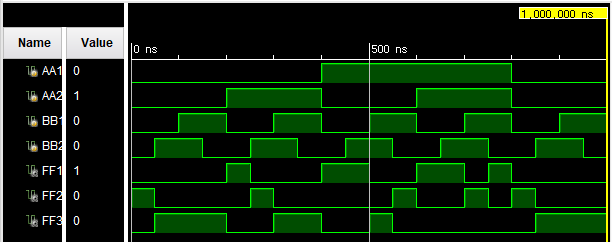
자동 생성된 설명

twobitcompare.v

텍스트이(가) 표시된 사진

자동 생성된 설명

Twobitcompare\_tb.v



Simulation

2-bit binary comparator를 보자. 2-bit binary comparator는 1bit comparator를 확장하여 두 자리에 대해 비교하는 비교기를 의미한다. 따라서 input으로 A1A2 두자리와 B1B2두자리 bit가 들어가고 output으로느 F1,F2,F3가 있다. 여기서 F1은 두자리를 비교하여 A1A2 > B1B2이면 1을 출력하고 F2는 두 input이 같을 때 1을 출력한다. F3는 B1B2 > A1A2일 때 1을 출력한다. 이를 진리표로 작성하면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In A1** | **In A2** | **In B1** | **In B2** | **Out F1** | **Out F2** | **Out F3** |
| **0** | **0** | **0** | **0** | **0** | **1** | **0** |
| **0** | **0** | **0** | **1** | **0** | **0** | **1** |
| **0** | **0** | **1** | **0** | **0** | **0** | **1** |
| **0** | **0** | **1** | **1** | **0** | **0** | **1** |
| **0** | **1** | **0** | **0** | **1** | **0** | **0** |
| **0** | **1** | **0** | **1** | **0** | **1** | **0** |
| **0** | **1** | **1** | **0** | **0** | **0** | **1** |
| **0** | **1** | **1** | **1** | **0** | **0** | **1** |
| **1** | **0** | **0** | **0** | **1** | **0** | **0** |
| **1** | **0** | **0** | **1** | **1** | **0** | **0** |
| **1** | **0** | **1** | **0** | **0** | **1** | **0** |
| **1** | **0** | **1** | **1** | **0** | **0** | **1** |
| **1** | **1** | **0** | **0** | **1** | **0** | **0** |
| **1** | **1** | **0** | **1** | **1** | **0** | **0** |
| **1** | **1** | **1** | **0** | **1** | **0** | **0** |
| **1** | **1** | **1** | **1** | **0** | **1** | **0** |

각 output F1,F2,F3에 대해 boolean식을 구하기 위하여 K-map을 아래 작성해보자.

-F1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A1A2  B1B2 | 00 | 01 | 11 | 10 |
| 00 | **0** | **1** | **1** | **1** |
| 01 | **0** | **0** | **1** | **1** |
| 11 | **0** | **0** | **0** | **0** |
| 10 | **0** | **0** | **1** | **0** |

F1의 카르노 맵에서 prime implicant를 묶어 Boolean 식으로 작성하자. 오른쪽 위 4개의 1을 묶고 3열 1행 4행, 1행 2열 3열을 묶는다. 계산하면 F1=(A1&~B1) | (A2&~B1&~B2) | (A1&A2&~B2)이다.

-F2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A1A2  B1B2 | 00 | 01 | 11 | 10 |
| 00 | **1** | **0** | **0** | **0** |
| 01 | **0** | **1** | **0** | **0** |
| 11 | **0** | **0** | **1** | **0** |
| 10 | **0** | **0** | **0** | **1** |

F2의 카르노 맵에서 implicant를 묶어 Boolean 식으로 작성하자. F2는 각 1을 각각 묶어야 한다. 묶고 계산하면 F2= ~((~A1&B1)|(A1&~B1)) & ~((~A2&B2) | (A2&~B2)) 이다. F2는 1bit compare에서 두 bit가 같은 것을 XOR로 구성했는데 2bit또한 XOR을 각각 자리의 bit에 걸어 구성하는 것을 쉽게 생각할 수 있다.

-F3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A1A2  B1B2 | 00 | 01 | 11 | 10 |
| 00 | **0** | **0** | **0** | **0** |
| 01 | **1** | **0** | **0** | **0** |
| 11 | **1** | **1** | **0** | **1** |
| 10 | **1** | **1** | **0** | **0** |

F1의 카르노 맵에서 prime implicant를 묶어 Boolean 식으로 작성하자. 왼쪽 밑 4개의 1을 묶고 3행 1열 4열, 1열 2행 3행을 묶는다. 계산하면 F3 = (~A1&B1)|(~A2&B1&B2)|(~A1&~A2&B2)이다.

위의 Boolean 식을 verilog에 작성하여 simulation결과를 truth table과 비교해보자. 맨 위 사진과 simulation결과를 보면 카르노 맵을 그리기 위한 truth table과 일치하는 것을 확인할 수 있다.

1. **결과 검토 및 논의사항.**

7주차에서는 2bit 비교기와 parity bit generator와 checker를 만들고 회로를 구성했다. 이를 만족하도록 직접 구성한 truth table과 그 table을 토대로 카르노 맵 작성 및 Boolean 식을 유도했다. 이를 Verilog 코드를 통해 구현하고 simulation을 확인한 결과 직접 구현한 truth table과 boolean식이 simulation결과와 일치하는 것을 확인할 수 있었다. 이를 FPGA와 연동하여 실습 진행 중 LED를 통해 verilog의 코드가 잘 작동하는지 확인했다.

1. **추가 이론 조사 및 작성**

FPGA의 동작을 이번 대면 실습 때 처음 실행했다. Verilog에서 구현한 input, output변수들을 FPGA의 pin과 port로 연동해야 FPGA에서의 작동을 확인할 수 있다. verilog에서 source코드를 구성한 후, constraint source를 추가하여 pin과 port 연결을 진행한다. 이 후, synthesis와 implementation을 open하고 Device와 연결하면 FPGA에서 동작을 확인할 수 있다. 연결한 pin에 대해 버튼을 통해 0과 1로 input을 조정할 수 있으며 output은 LED pin으로 연결하여 output이 1일 때 LED에 불이 들어온다.