Lab #07: 4-bit Adder/Subtractor

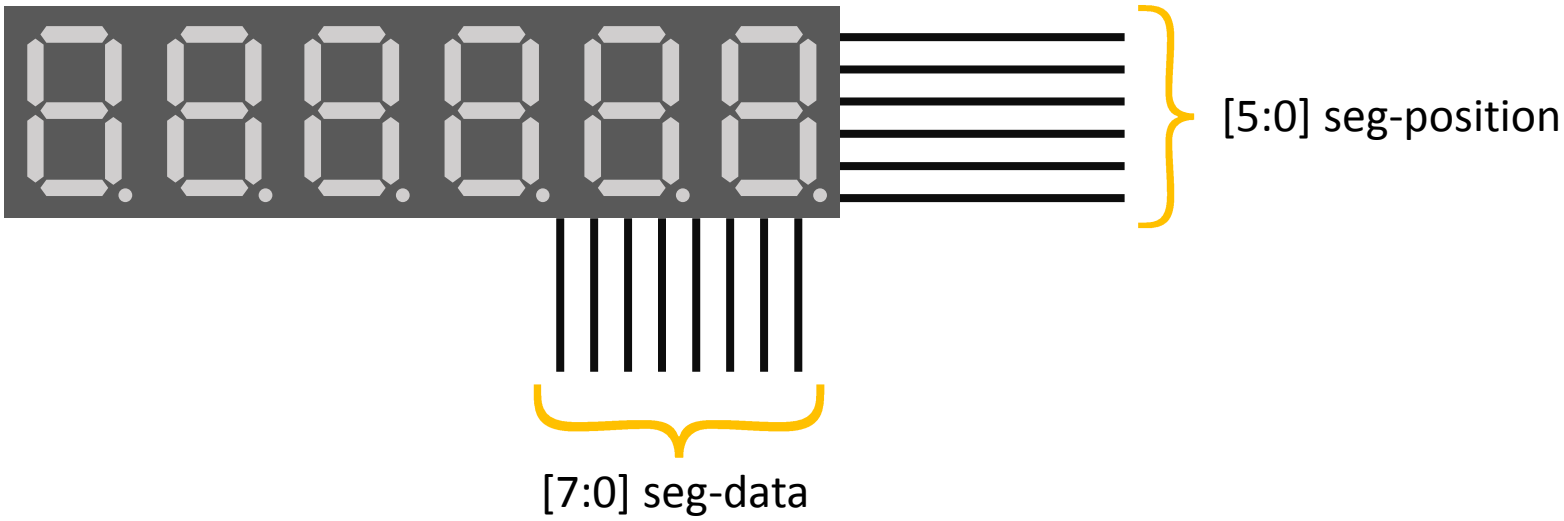
4-bit 가(감)산기 설계하기

연세대학교 컴퓨터 과학과 디지털 논리회로 실습

Computer Science Department of
Yonsei University

Review: 7-Segments

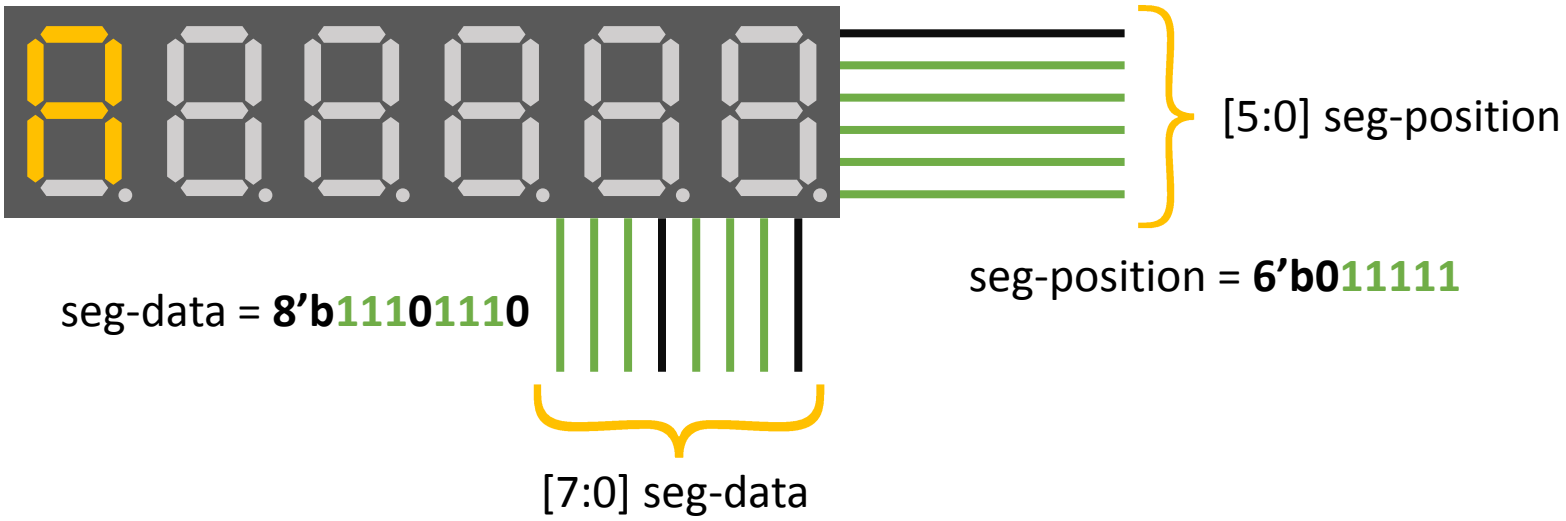
- 7-Segment 표시 장비 복습 / 원리 이해
  - 7-Segment를 동작시키는데 필요한 신호들:
    - 7개의 Segment와 Dot-point를 밝히는데 필요한 정보 ~ 8-bit
    - 6 자리의 Segments 중, 자릿수를 결정하는데 필요한 정보 ~ 6-bit
    - 총 14-bit가 사용된다. (하지만, 6자리 X 8개 segment = 48-bit가 필요하지 않을까?)



  - 이러한 7-Segment 장치에서 일반적인 설계 방법으로, 여섯 자리 숫자가 각기 다른 수를 표현 할 수 있을까?

Review

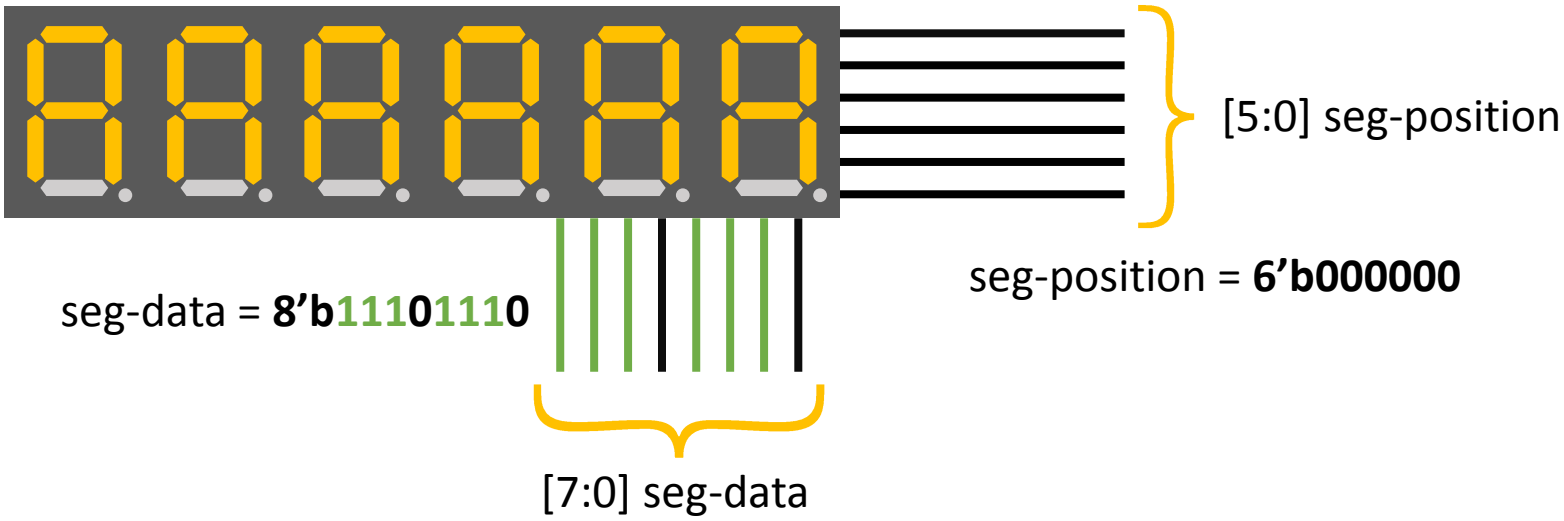
- 7-Segment 표시 장비 복습 / 원리 이해
  - 7-Segment를 동작시키는데 필요한 신호들:
    - 7개의 Segment와 Dot-point를 밝히는데 필요한 정보 ~ 8-bit
    - 6 자리의 Segments 중, 자릿수를 결정하는데 필요한 정보 ~ 6-bit
    - 총 14-bit가 사용된다. (하지만, 6자리 X 8개 segment = 48-bit가 필요하지 않을까?)



  - 이러한 7-Segment 장치에서 일반적인 설계 방법으로, 여섯 자리 숫자가 각기 다른 수를 표현 할 수 있을까?

Review

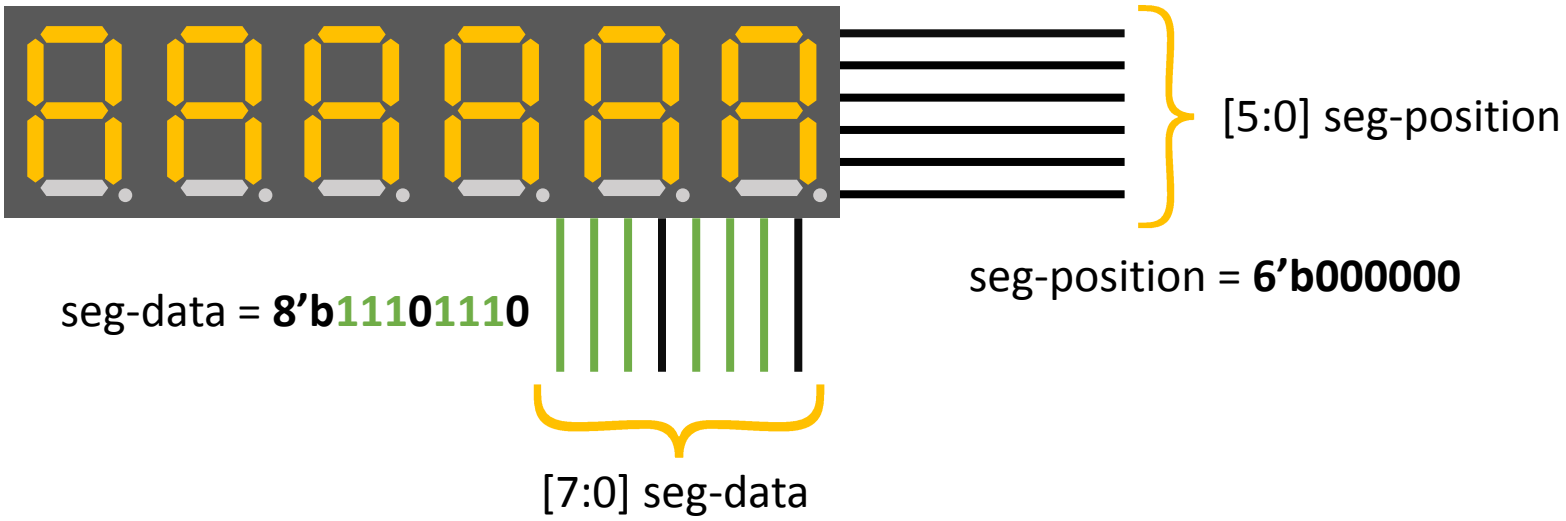
- 7-Segment 표시 장비 복습 / 원리 이해
  - 7-Segment를 동작시키는데 필요한 신호들:
    - 7개의 Segment와 Dot-point를 밝히는데 필요한 정보 ~ 8-bit
    - 6 자리의 Segments 중, 자릿수를 결정하는데 필요한 정보 ~ 6-bit
    - 총 14-bit가 사용된다. (하지만, 6자리 X 8개 segment = 48-bit가 필요하지 않을까?)



  - 이러한 7-Segment 장치에서 일반적인 설계 방법으로, 여섯 자리 숫자가 각기 다른 수를 표현 할 수 있을까?

Review

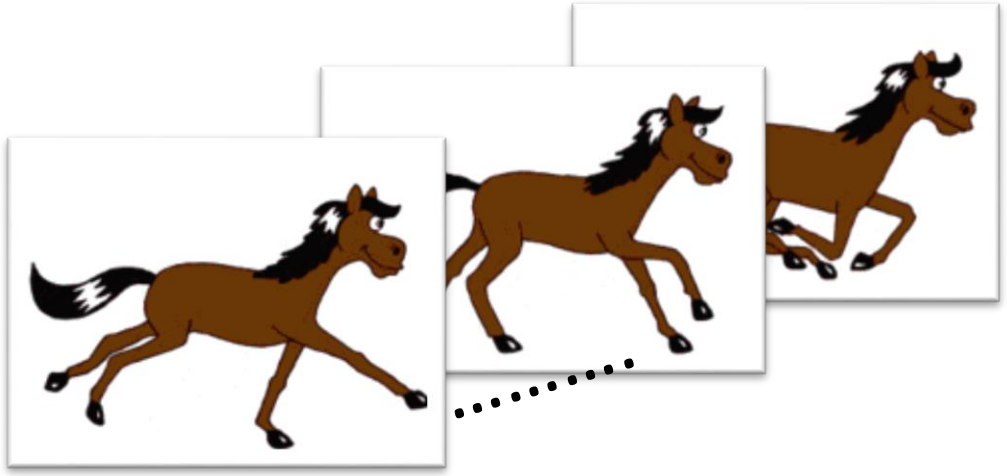
- 7-Segment 표시 장비 복습 / 원리 이해
  - 7-Segment를 동작시키는데 필요한 신호들:
    - 7개의 Segment와 Dot-point를 밝히는데 필요한 정보 ~ 8-bit
    - 6 자리의 Segments 중, 자릿수를 결정하는데 필요한 정보 ~ 6-bit
    - 총 14-bit가 사용된다. (하지만, 6자리 X 8개 segment = 48-bit가 필요하지 않을까?)



  - 그렇다면, 각기 다른 여섯 자리 수를 표현하기 위해서는 어떻게 해야 할까?

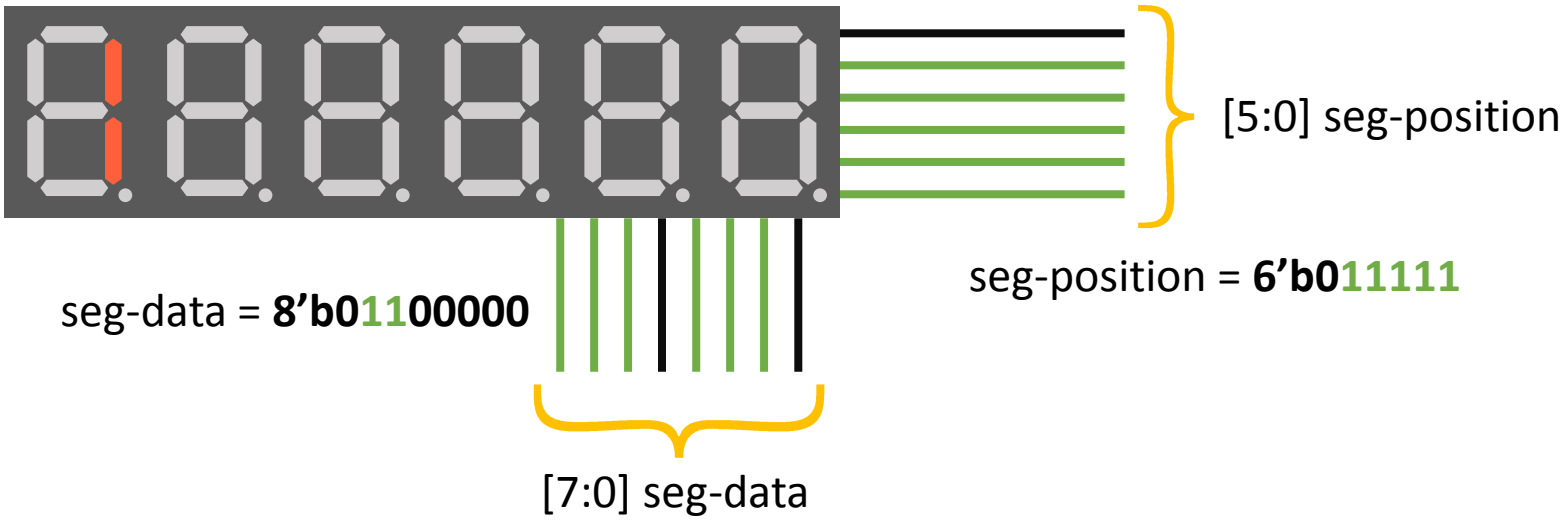
Review

- IDEA: 눈의 잔상 효과를 이용해보자!



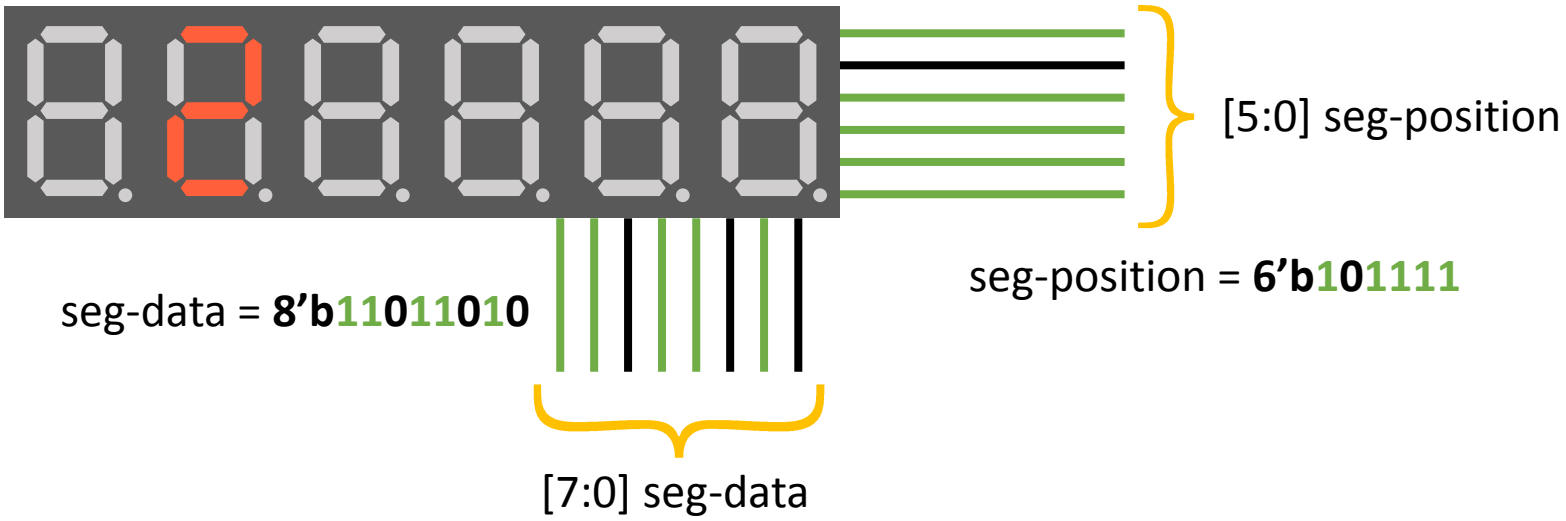
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기



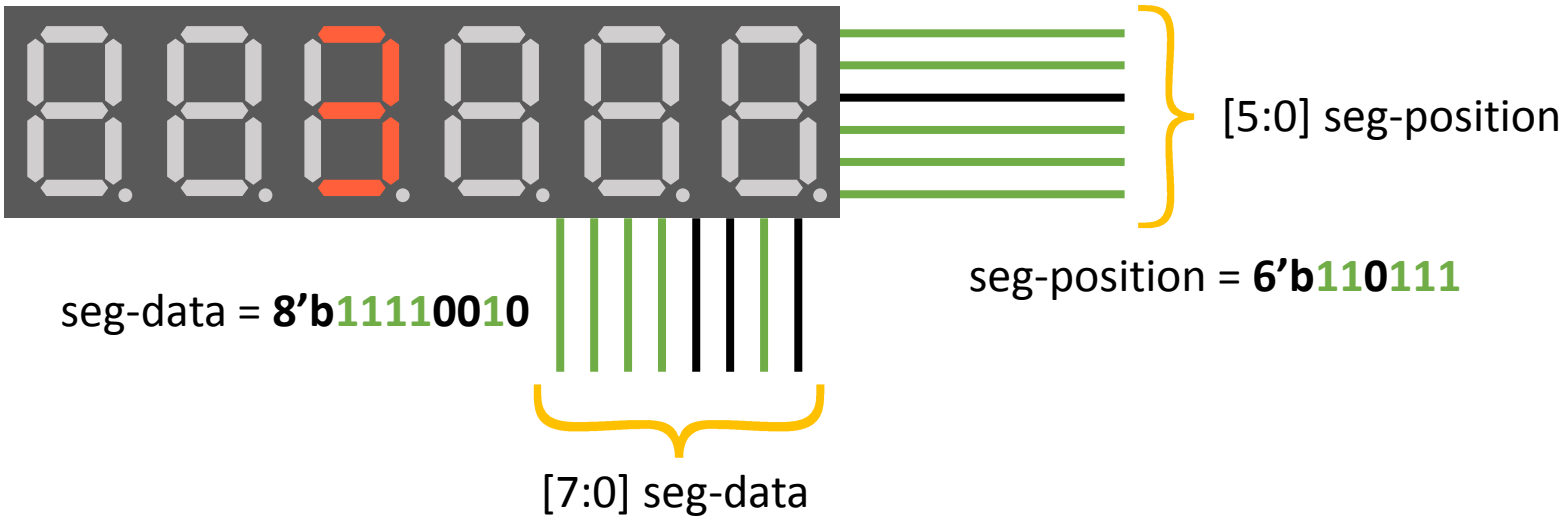
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기



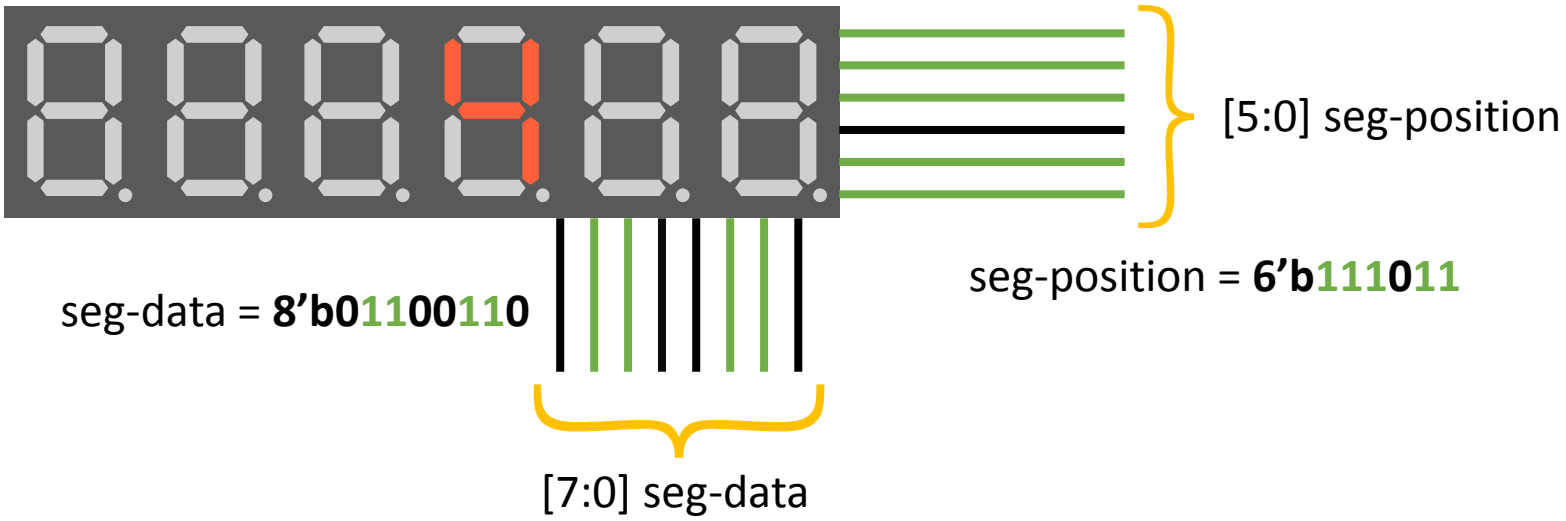
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기



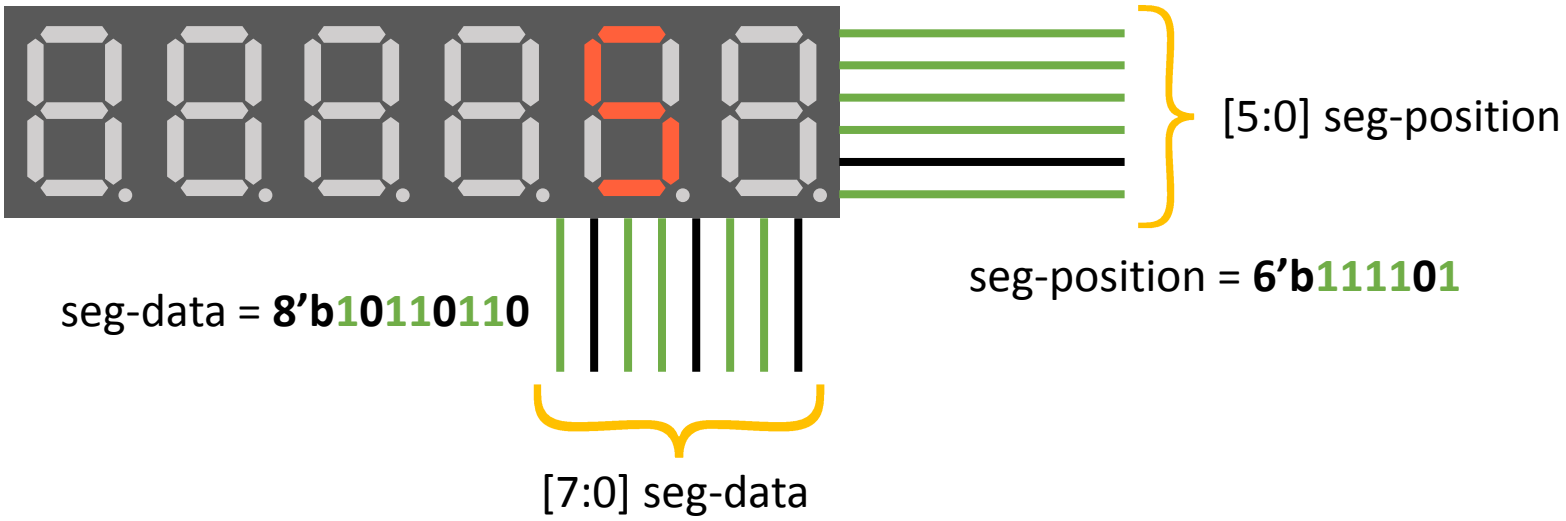
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기



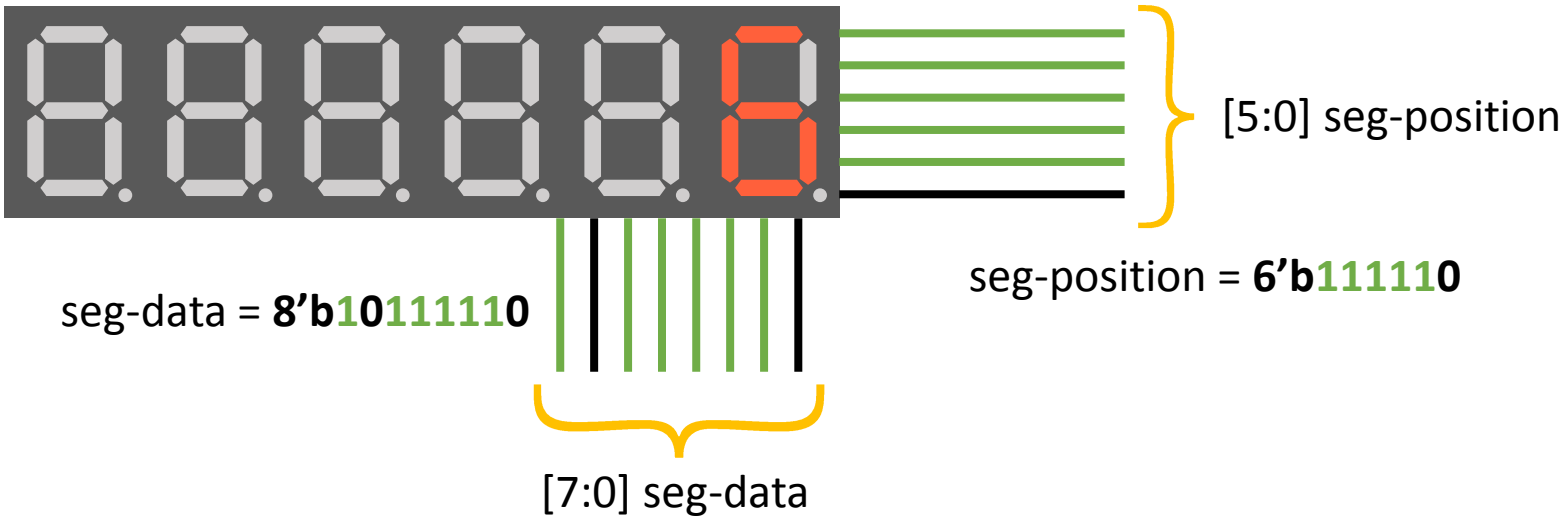
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기



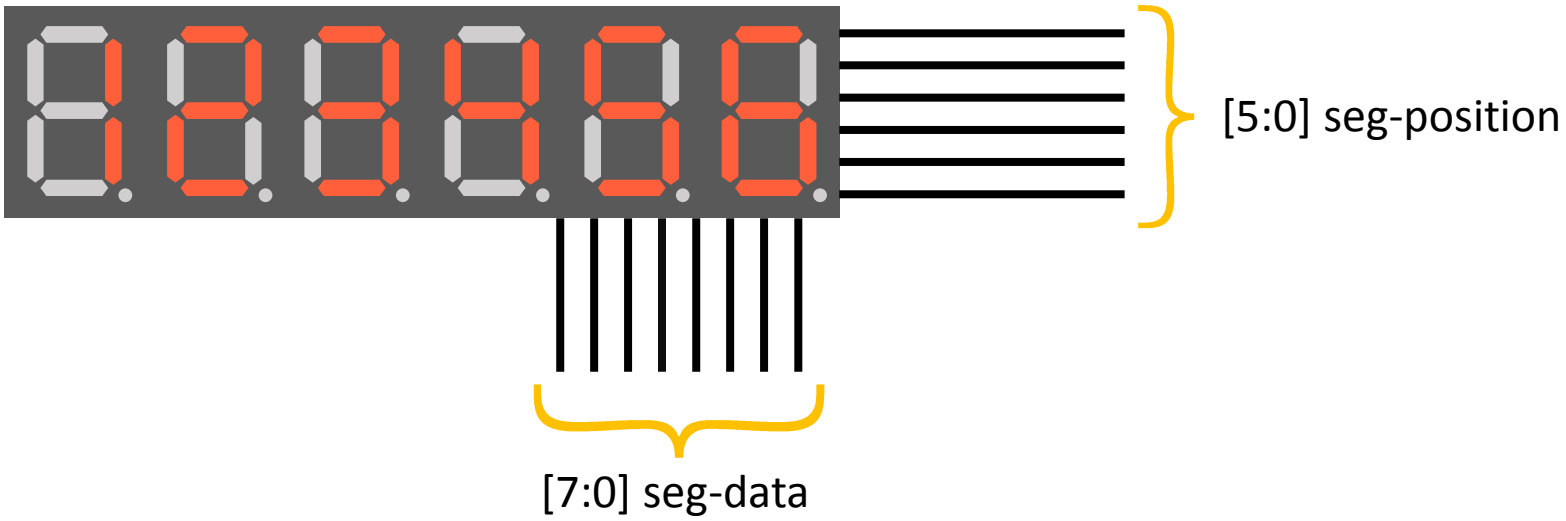
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기



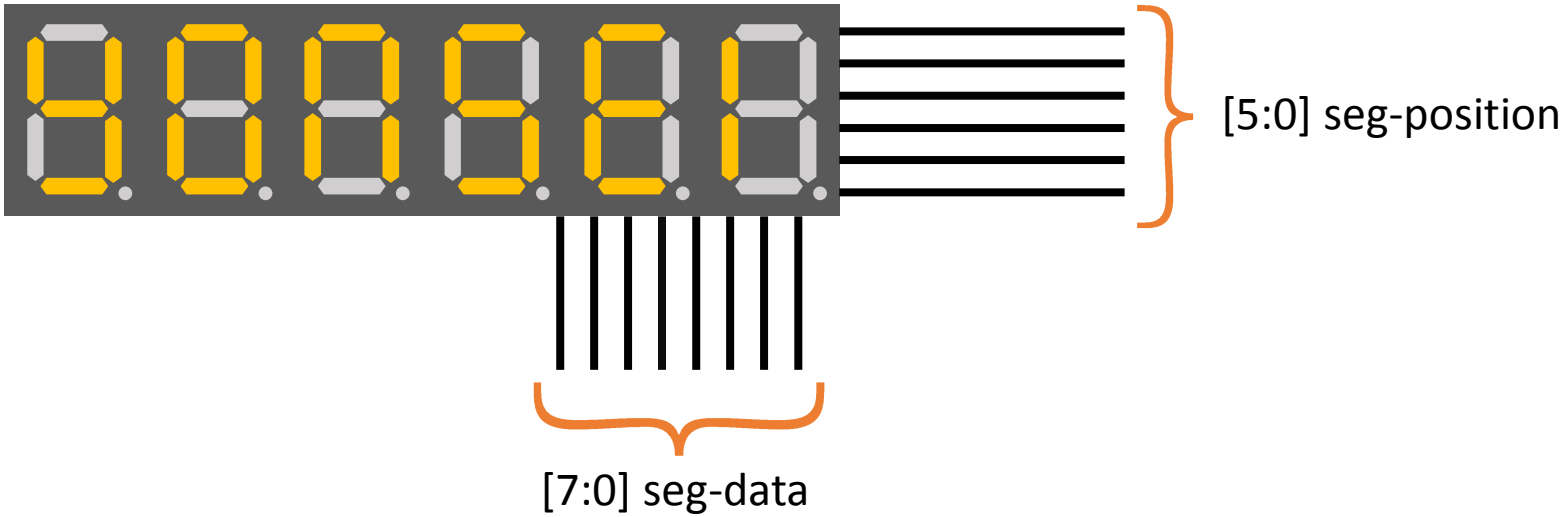
Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기
  - 앞의 6단계를 짧은 시간 동안, 매우 빠르게 반복하면 아래와 같이 보인다!
  - 어느 정도의 짧은 간격으로 할지는, 반복 작업을 통해 경험적으로 파악해본다.



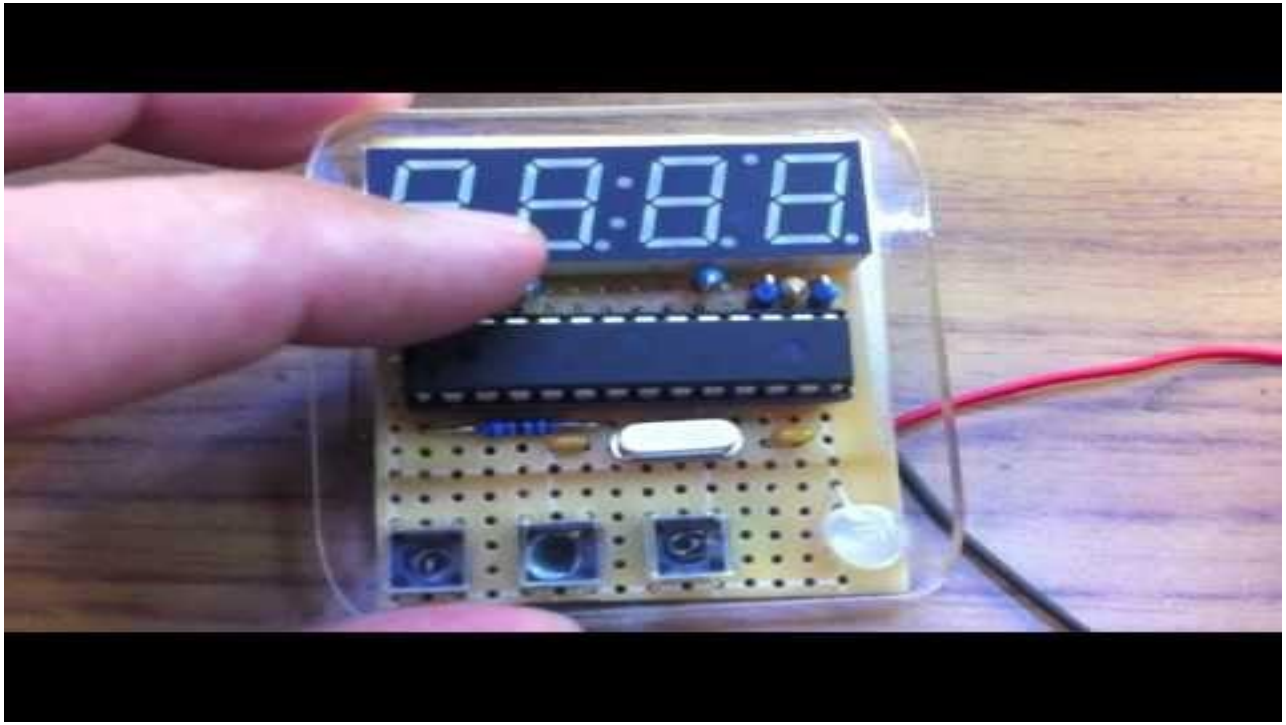
Review – YONSEI 전광판

- 7-Segment controller에 적용되는 원리를 이용한다.
- 6자릿수의 숫자를 표시하는 것과 조금 다른 것이 있다면,
  전광판은 일정 시간 간격에 따라 “YONSEI”라는 단어가 왼쪽으로 이동한다는 점.
- 따라서 좀 더 생각해 볼 것들이 있다.



Review

- 눈의 잔상 현상 / 뇌의 착각을 이용해서, 각기 다른 여섯 자리 수 표현하기
  - 앞의 6단계를 짧은 시간 동안, 매우 빠르게 반복하면 아래와 같이 보인다!
  - 어느 정도의 짧은 간격으로 할지는, 반복 작업을 통해 경험적으로 파악해본다.
  - (https://www.youtube.com/watch?v=R9kUadTXpzs)



Key-matrix

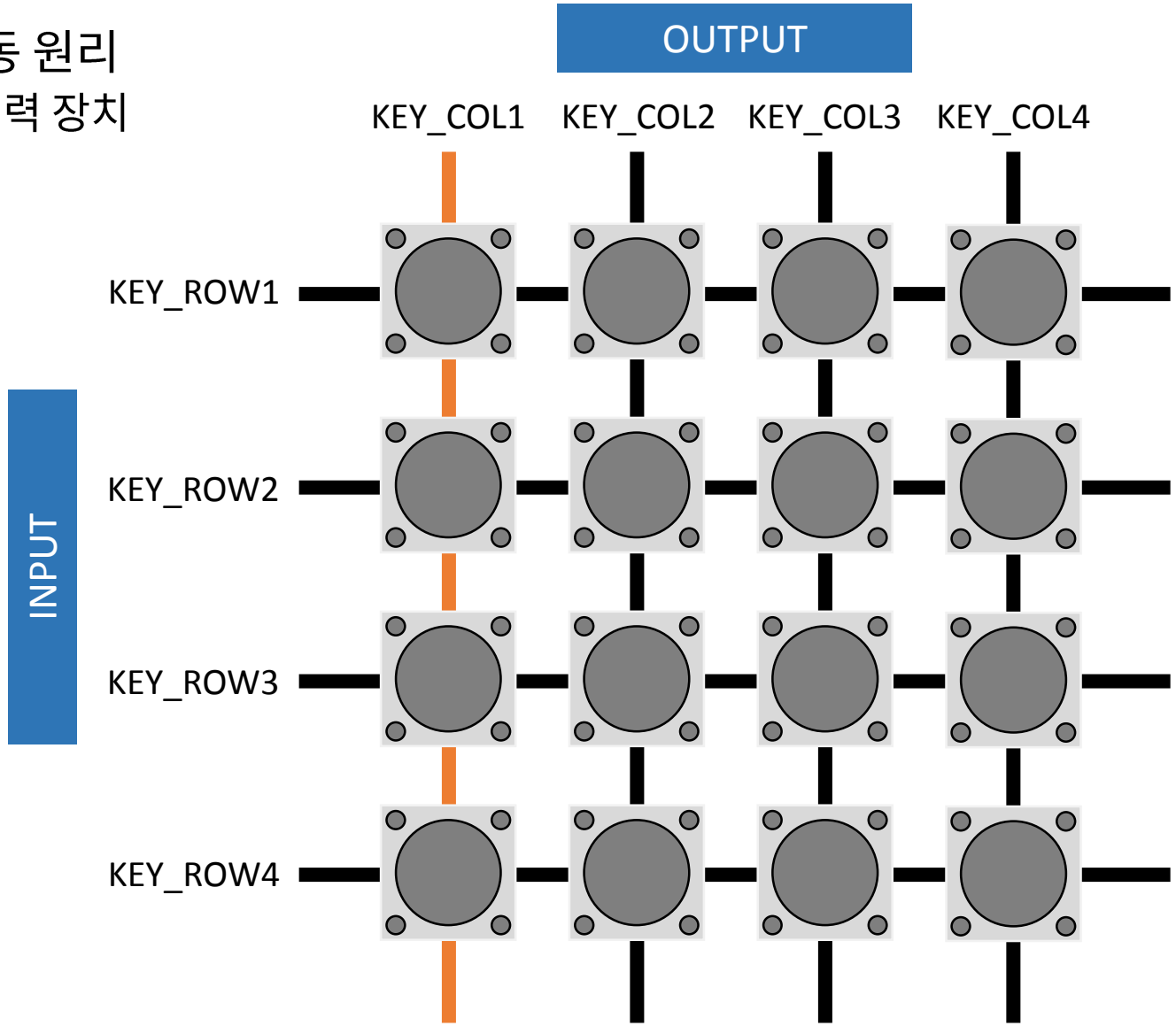
- Key-matrix?
  - 4x4의 key 입력 장치
    - FPGA 보드를 보면, KEY_COL1 ~ KEY_COL4 / KEY_ROW1 ~ KEY_ROW4라 적혀있다.
    - 문제는 key의 총 개수는 16개이다.
      (Q: key가 16개면, 이를 감지하는 wire도 16개여야 하지 않는가?)
    - 앞의 7-Segment 장비의 특성을 통해 유추해보면, 이 장비도
      '사람이 버튼을 누르는 간격은, 일정 수준 이상의 시간 간격을 두고 일어난다.'
      라는 점에 착안해서 wire의 수를 절감하는 방식을 택한 것으로 볼 수 있다.
    - KEY_COL1~4가 output wire, KEY_ROW1~4가 input wire이다.
      - 즉, KEY_COL의 1부터 4까지 번갈아 가며 신호를 주고,
        이 신호가 활성화 되어 있는 상태에서 사용자가 버튼을 누를 경우
        KEY_ROW를 통해 신호가 감지된다는 원리를 이용
      - 다음의 슬라이드를 통해서 확실하게 동작 원리를 이해해본다.

Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 1
KEY_COL2 = 0
KEY_COL3 = 0
KEY_COL4 = 0

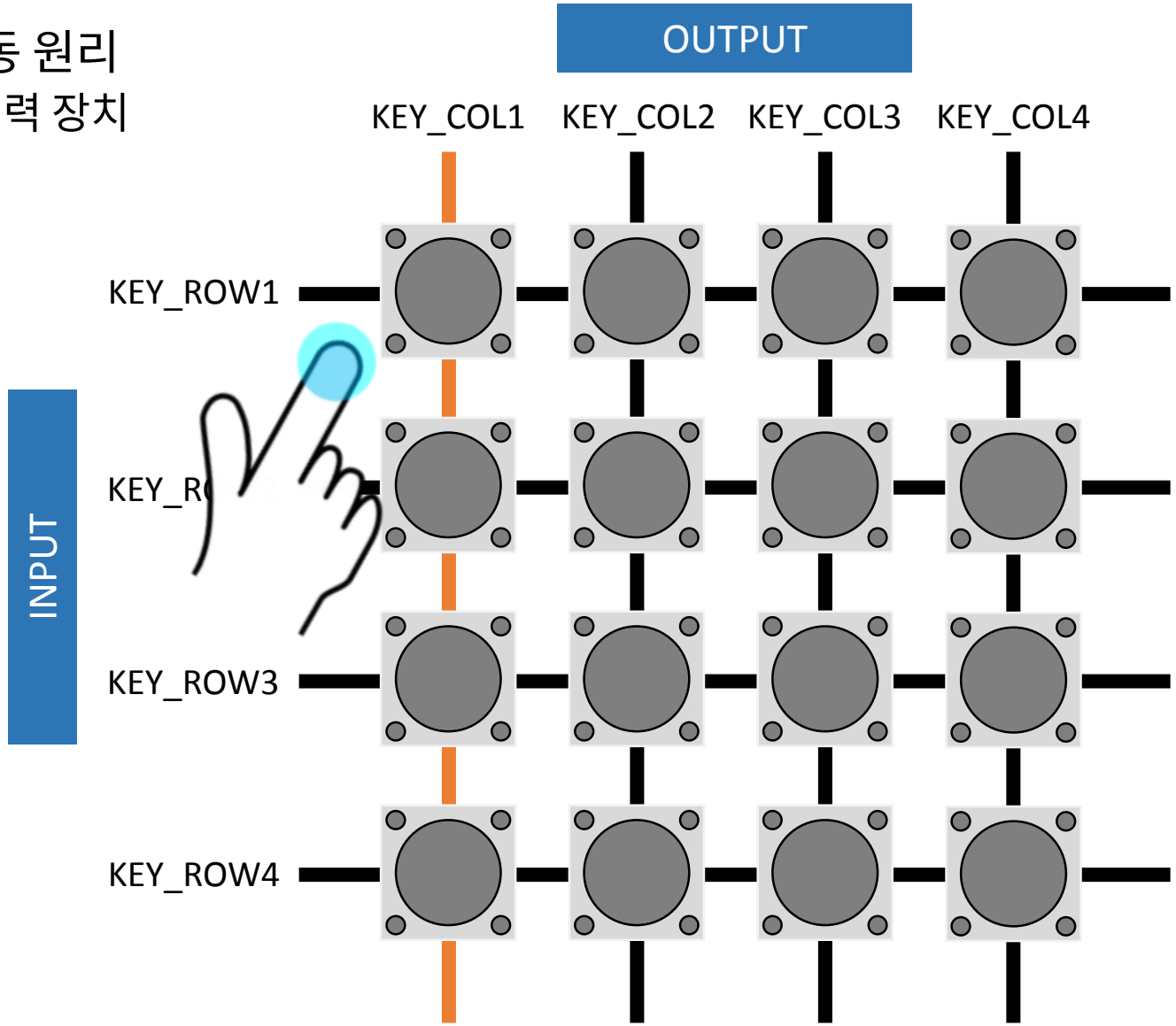


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 1
KEY_COL2 = 0
KEY_COL3 = 0
KEY_COL4 = 0

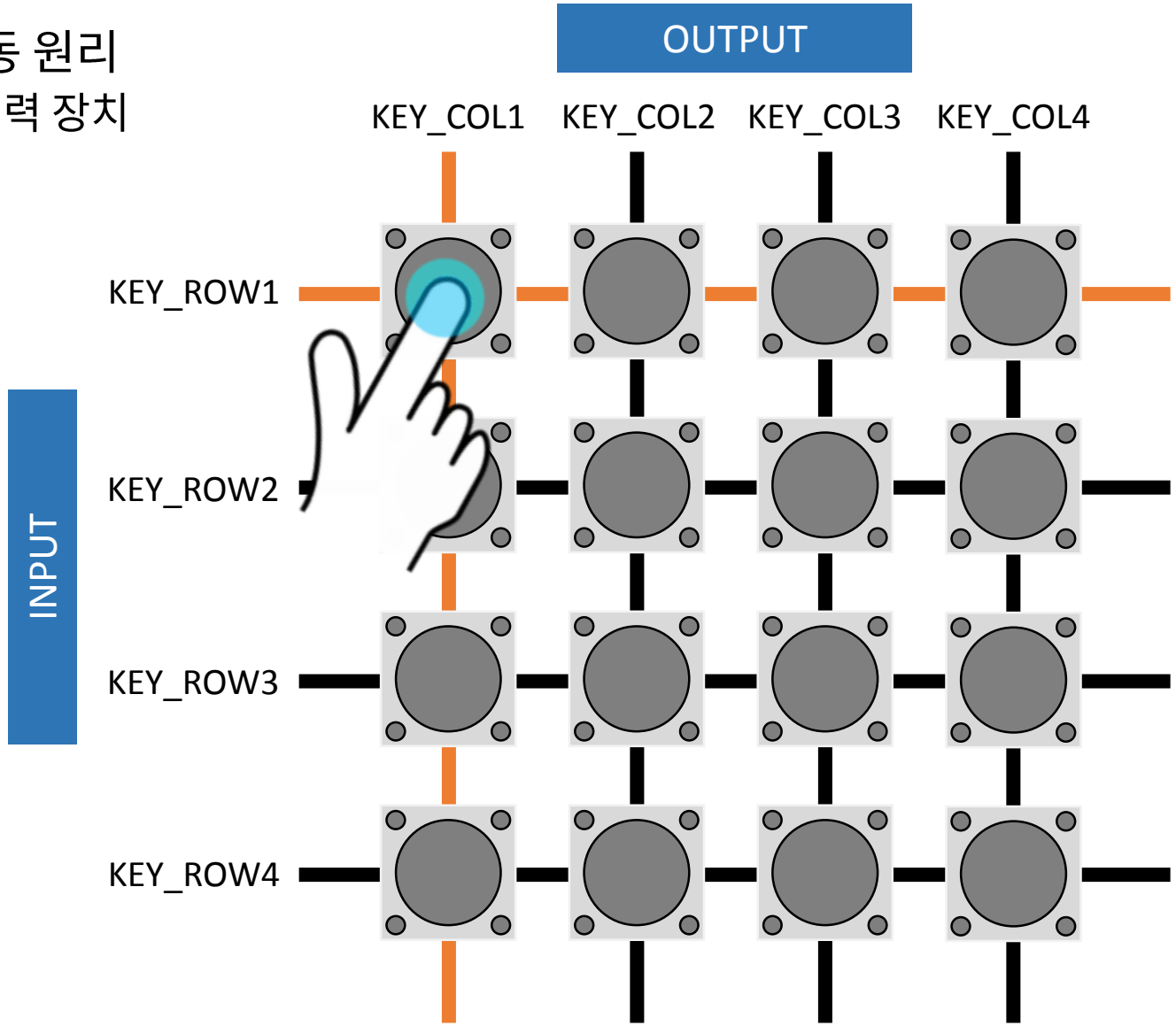


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

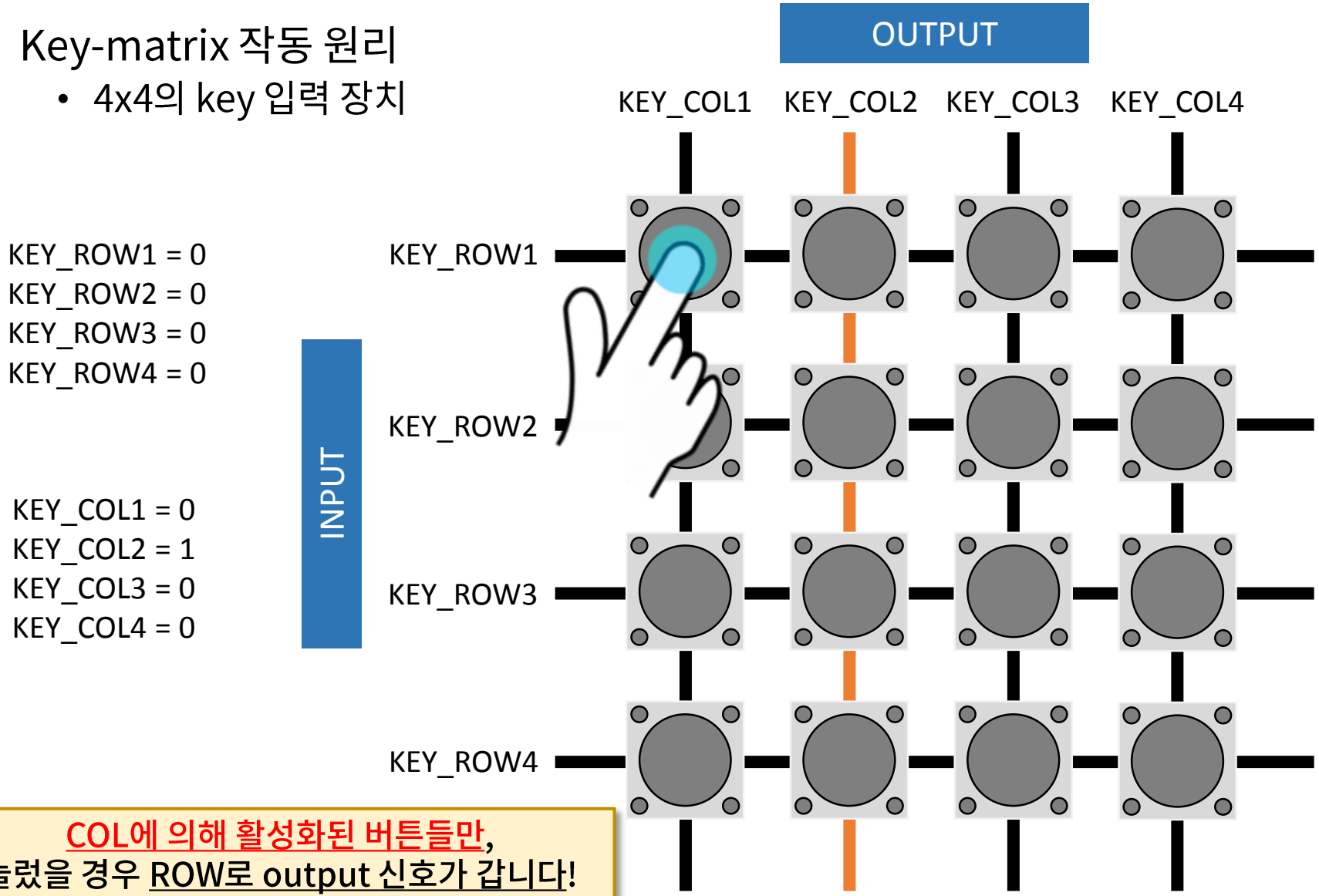
KEY_ROW1 = 1
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 1
KEY_COL2 = 0
KEY_COL3 = 0
KEY_COL4 = 0



Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치



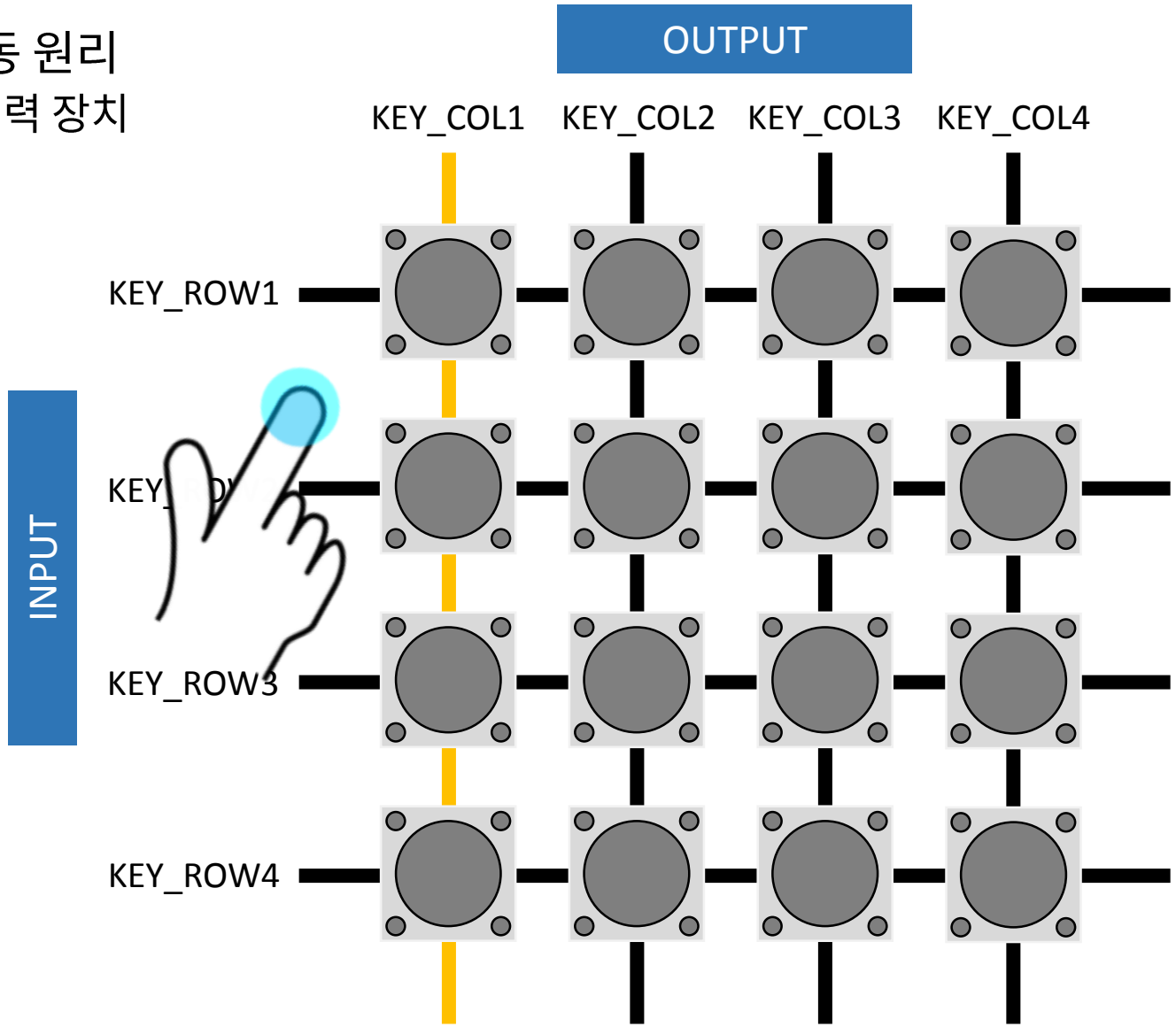
COL에 의해 활성화된 버튼들만,
눌렀을 경우 ROW로 output 신호가 갑니다!

Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 1
KEY_COL2 = 0
KEY_COL3 = 0
KEY_COL4 = 0

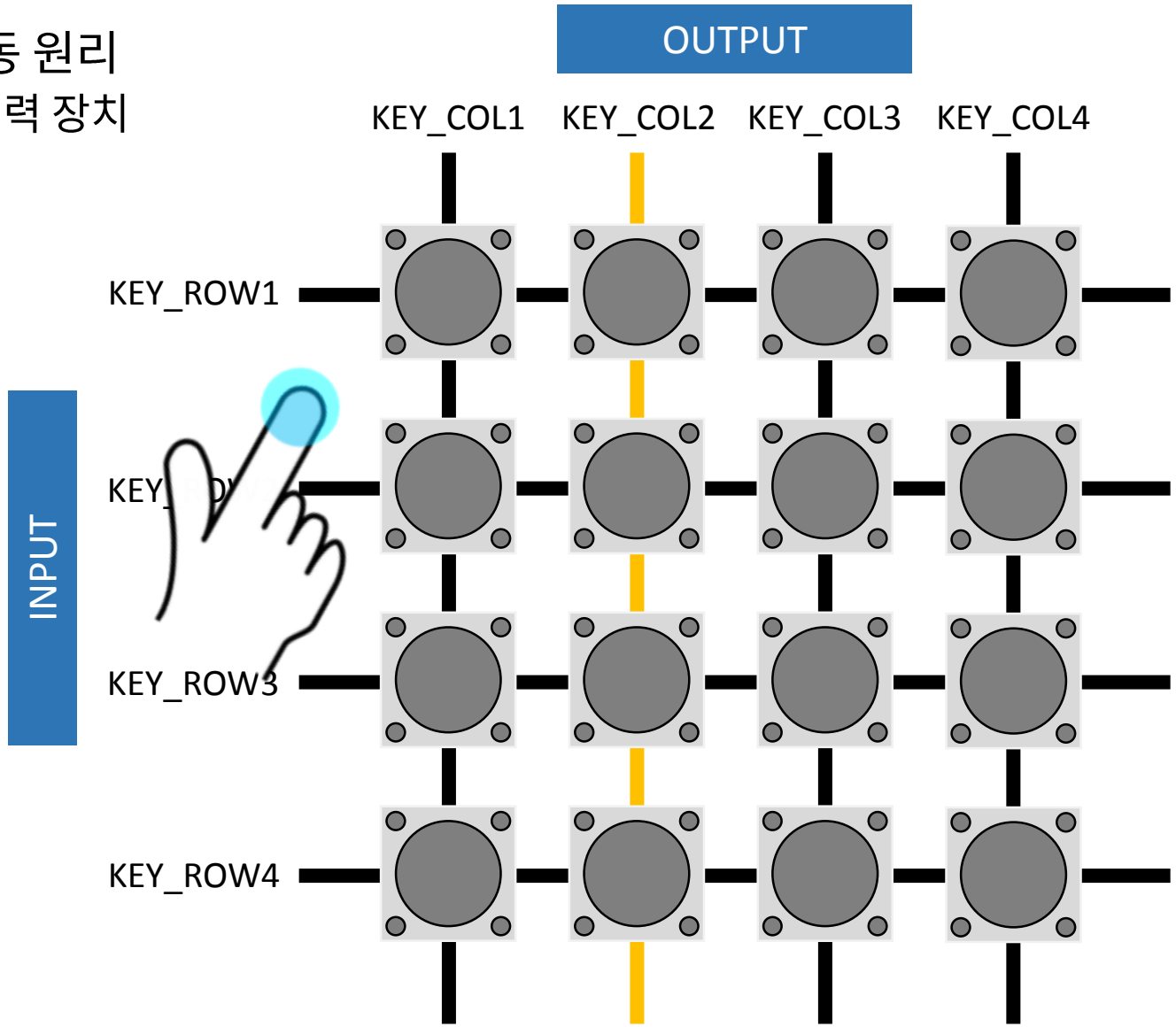


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 0
KEY_COL2 = 1
KEY_COL3 = 0
KEY_COL4 = 0

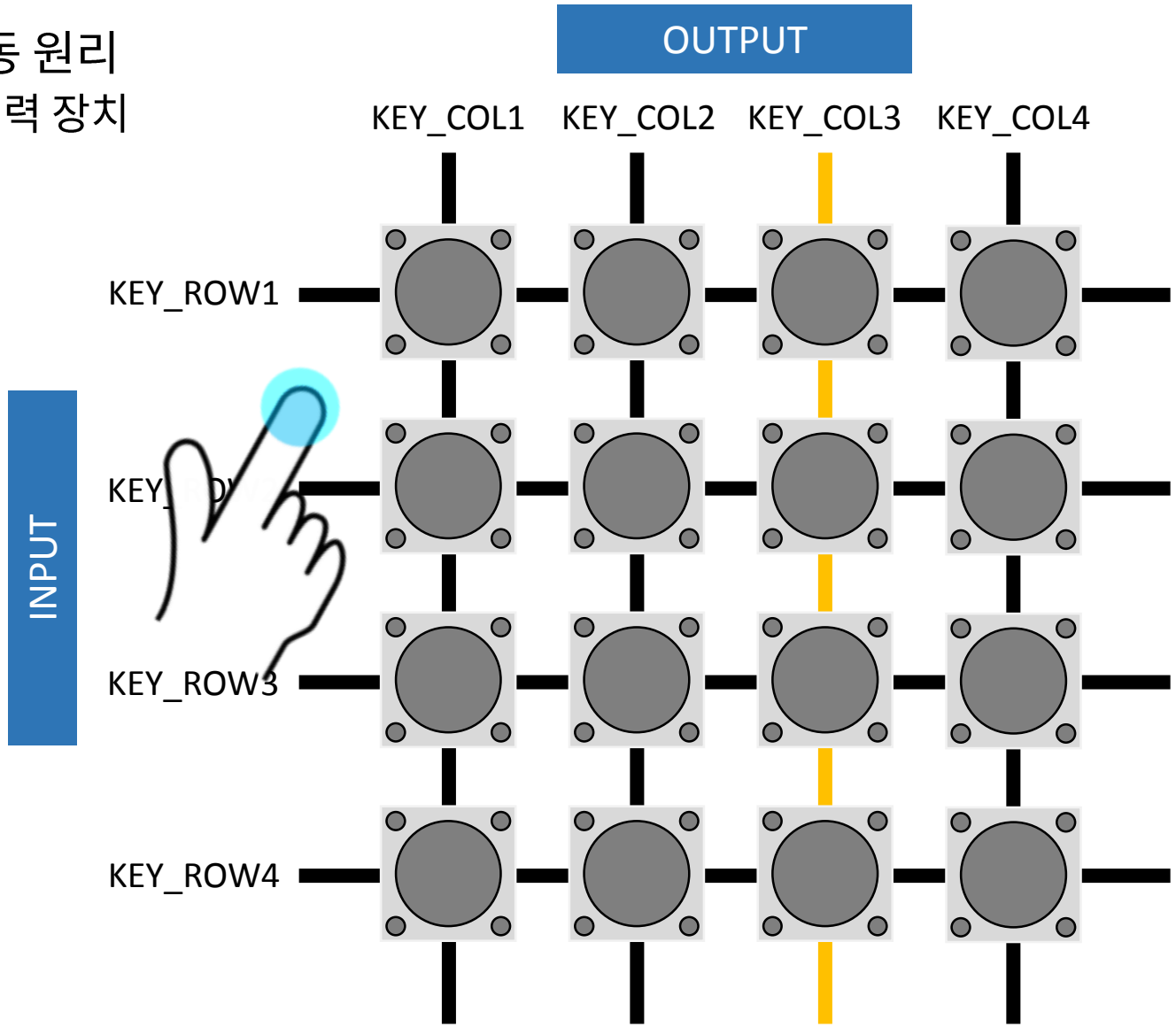


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

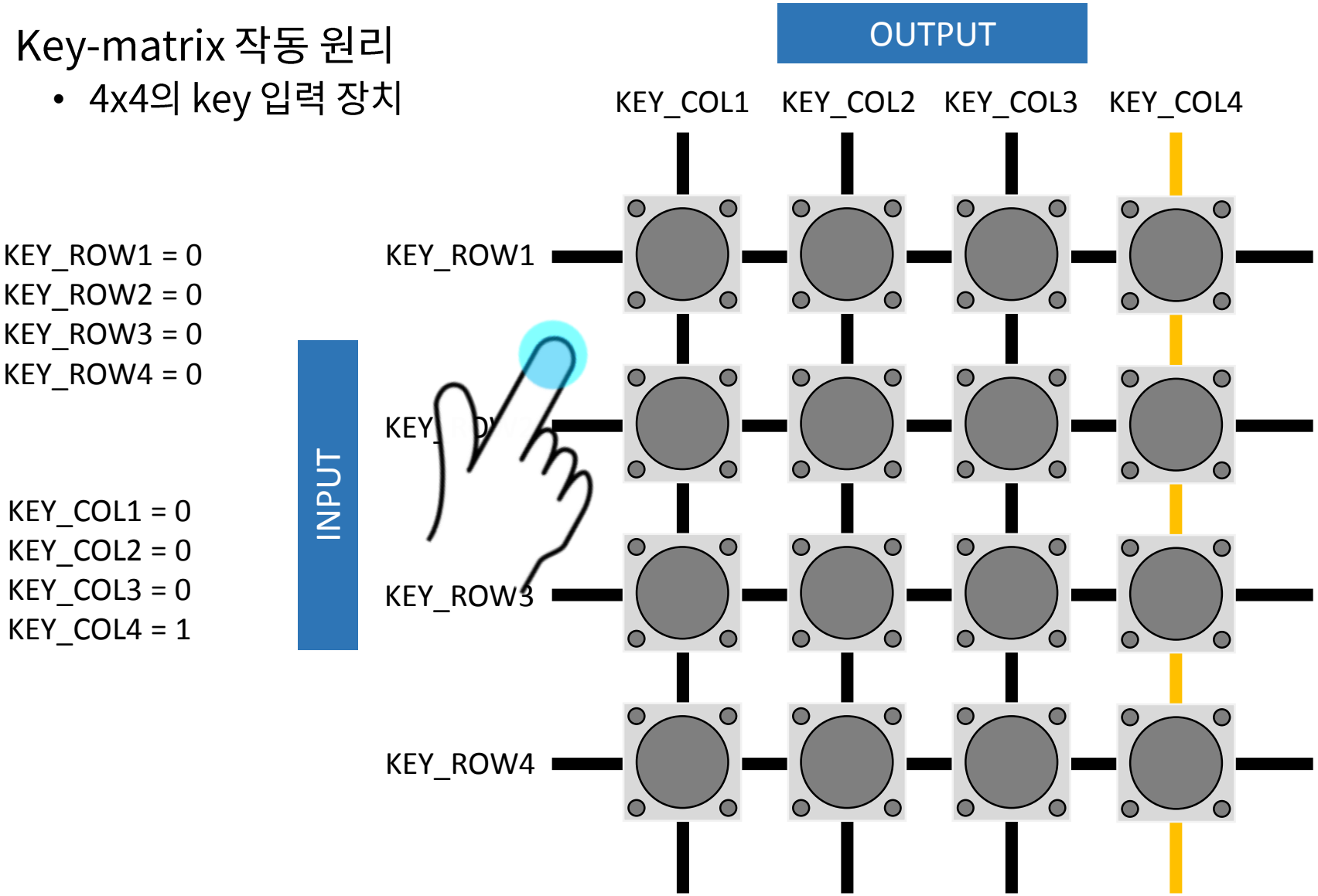
KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 0
KEY_COL2 = 0
KEY_COL3 = 1
KEY_COL4 = 0



Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

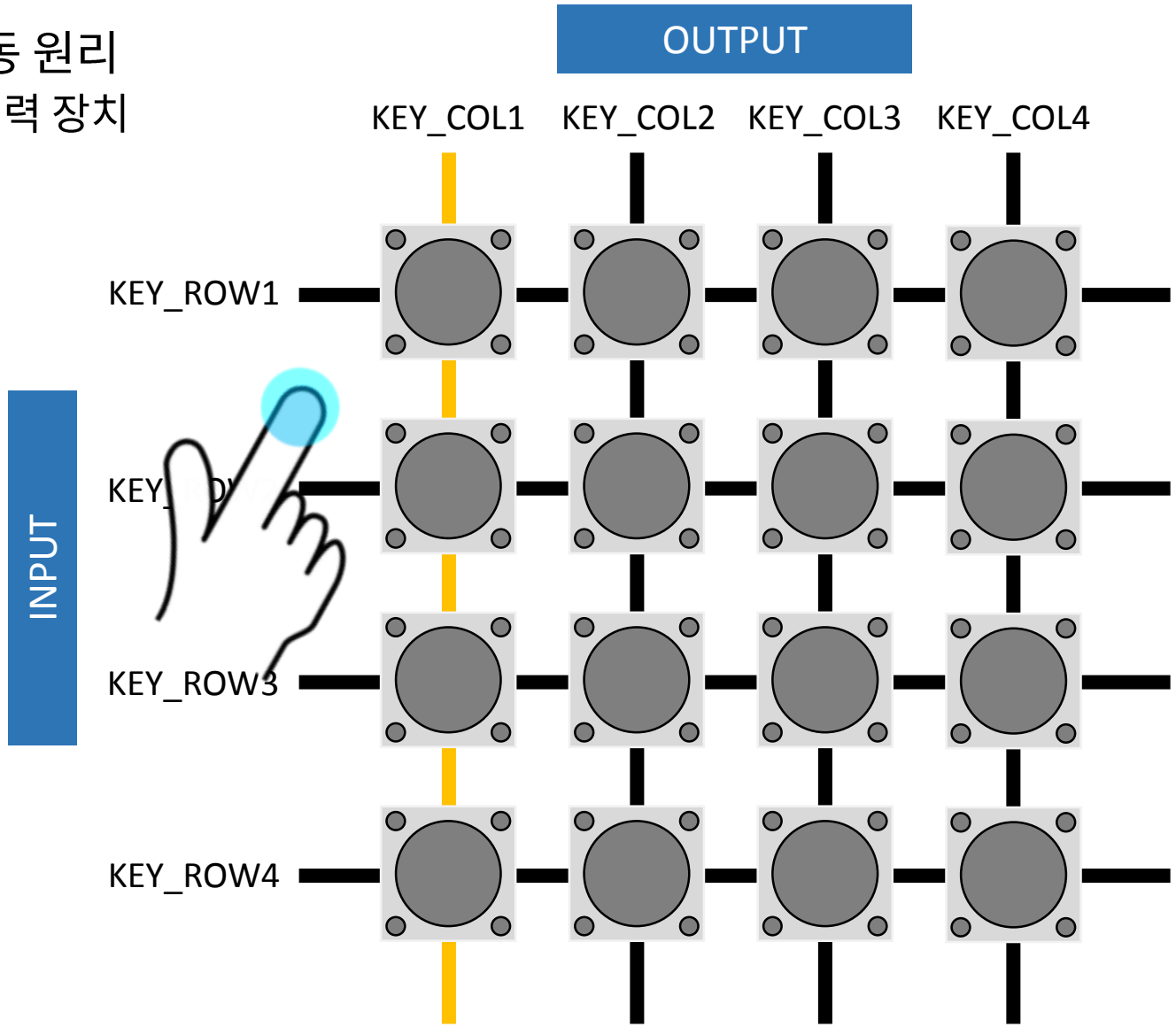


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 1
KEY_COL2 = 0
KEY_COL3 = 0
KEY_COL4 = 0

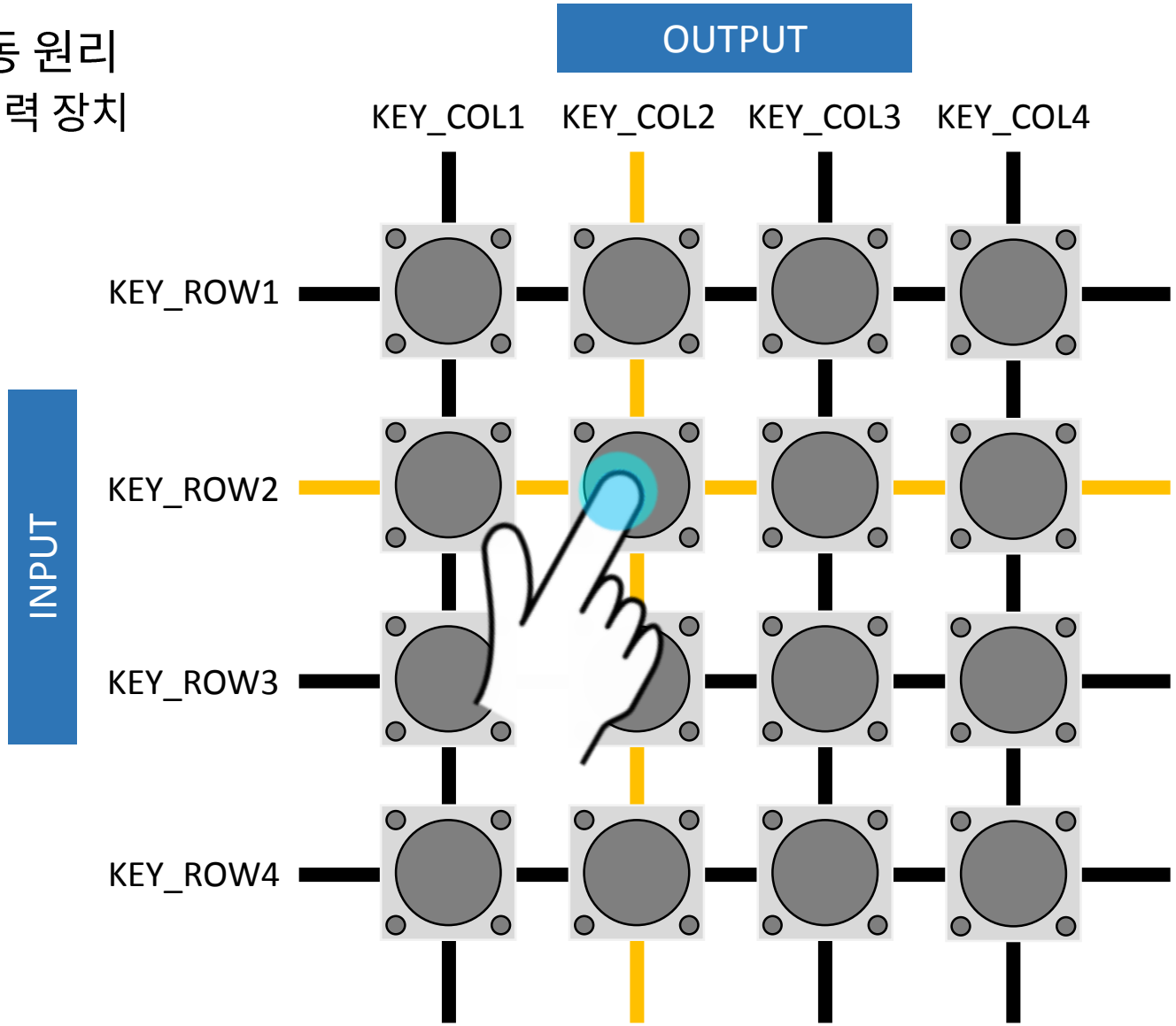


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

KEY_ROW1 = 0
KEY_ROW2 = 1
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 0
KEY_COL2 = 1
KEY_COL3 = 0
KEY_COL4 = 0

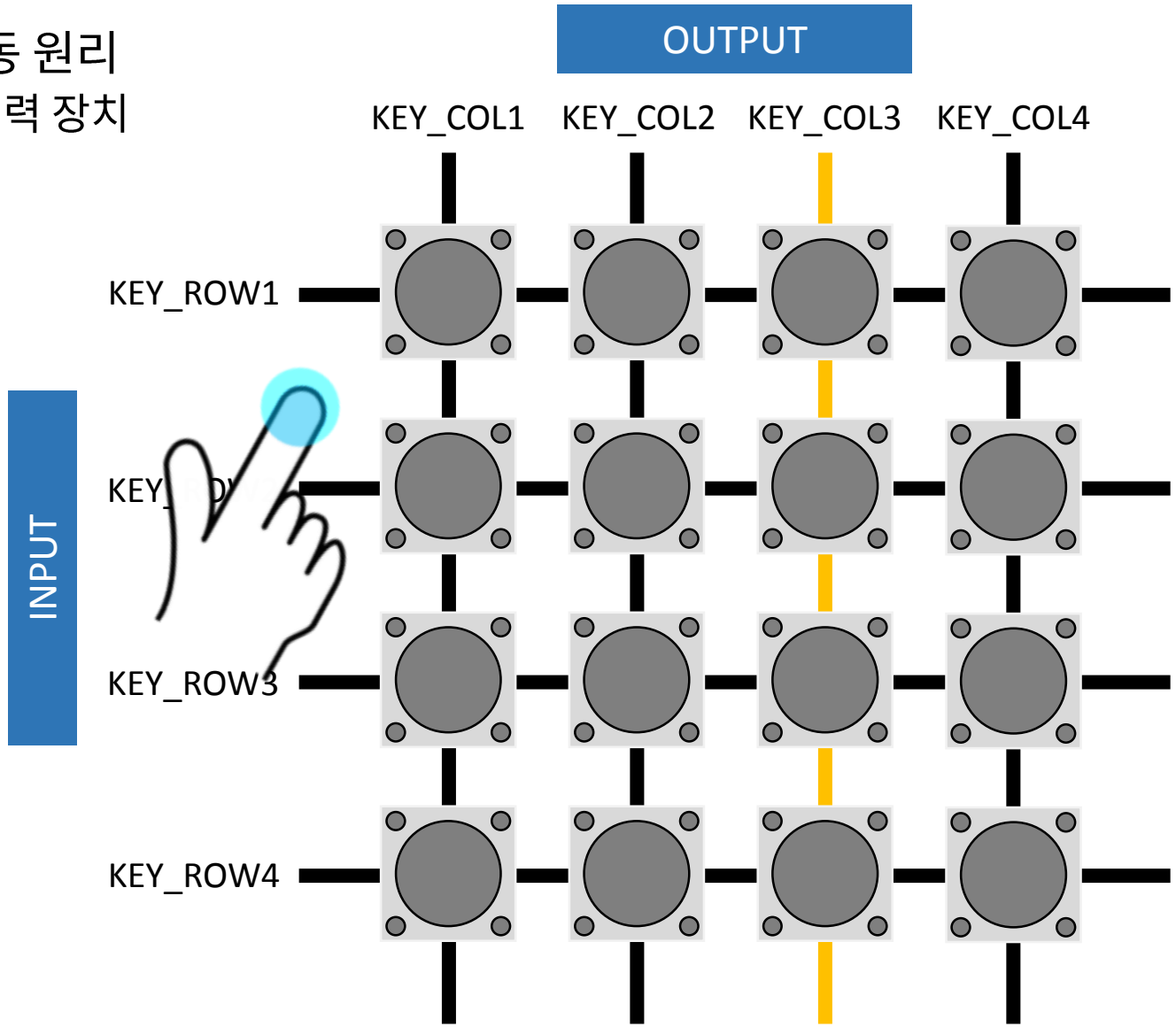


Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치

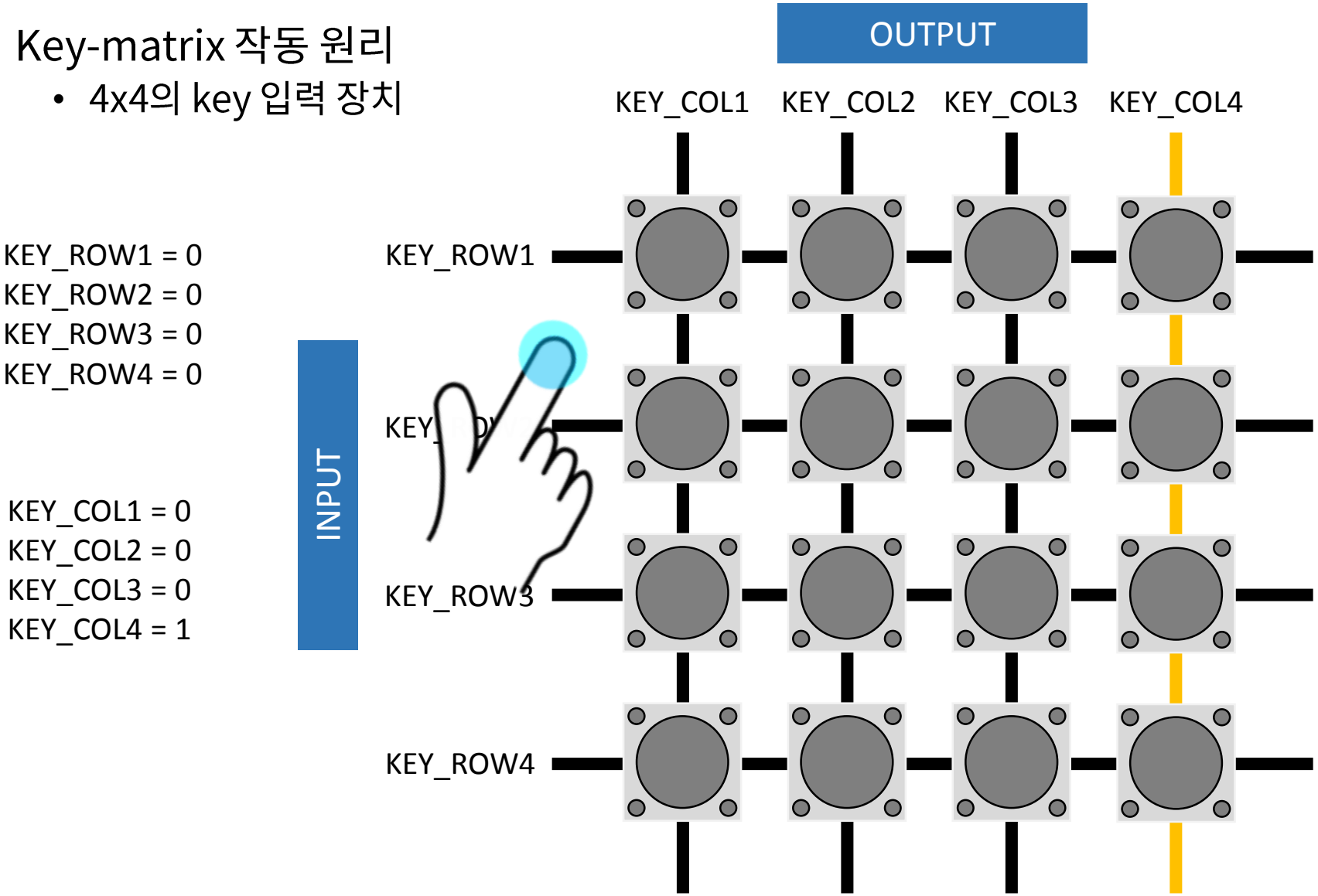
KEY_ROW1 = 0
KEY_ROW2 = 0
KEY_ROW3 = 0
KEY_ROW4 = 0

KEY_COL1 = 0
KEY_COL2 = 0
KEY_COL3 = 1
KEY_COL4 = 0



Key-matrix

- Key-matrix 작동 원리
  - 4x4의 key 입력 장치



Key-matrix

- Key-matrix 참조 코드 – 키패드를 눌러서 7-Segment에 표시하기

```
module keymatrix(clk, key_row, key_col, seg_pos, seg_data);
    input                   clk;
    input       [3:0]       key_row;

    output  [3:0]       key_col;
    output  [5:0]       seg_pos;
    output  [7:0]       seg_data;

    reg     [3:0]       key_col;
    reg     [7:0]       seg_data;
    reg     [5:0]       seg_pos;

    reg     [15:0]      cnt_for_6k;
    reg     [13:0]      cnt_for_device;
    reg                 clk_5kHz;

    reg     [1:0]       KEY_MATRIX_STATE;
    reg     [2:0]       FND_STATE;

    reg     [3:0]       key_value;


    // 64 * 5120 = 327680
    always @ (posedge clk)
    begin
        //  ---------|          |
        //  |           |_______|
        //  0           32          64
        if (cnt_for_6k == 0)                clk_5kHz = 1;
        else if (cnt_for_6k == 3200)    clk_5kHz = 0;
        cnt_for_6k = cnt_for_6k + 1;
        if (cnt_for_6k == 6400)         cnt_for_6k = 0;
    end
```

```
    always @ (negedge clk_5kHz)
    begin

        if      (KEY_MATRIX_STATE == 2'b00) key_col = 4'b1000;
        else if (KEY_MATRIX_STATE == 2'b01) key_col = 4'b0100;
        else if (KEY_MATRIX_STATE == 2'b10) key_col = 4'b0010;
        else if (KEY_MATRIX_STATE == 2'b11) key_col = 4'b0001;

        if (key_row == 4'b1000)
        begin
            if      (KEY_MATRIX_STATE == 2'b00) key_value = 1;
            else if (KEY_MATRIX_STATE == 2'b01) key_value = 10;
            else if (KEY_MATRIX_STATE == 2'b10) key_value = 3;
            else if (KEY_MATRIX_STATE == 2'b11) key_value = 2;
        end
        else if (key_row == 4'b0100)
        begin
            if      (KEY_MATRIX_STATE == 2'b00) key_value = 4;
            else if (KEY_MATRIX_STATE == 2'b01) key_value = 11;
            else if (KEY_MATRIX_STATE == 2'b10) key_value = 6;
            else if (KEY_MATRIX_STATE == 2'b11) key_value = 5;
        end
        else if (key_row == 4'b0010)
        begin
            if      (KEY_MATRIX_STATE == 2'b00) key_value = 7;
            else if (KEY_MATRIX_STATE == 2'b01) key_value = 12;
            else if (KEY_MATRIX_STATE == 2'b10) key_value = 9;
            else if (KEY_MATRIX_STATE == 2'b11) key_value = 8;
        end
        else if (key_row == 4'b0001)
        begin
            if      (KEY_MATRIX_STATE == 2'b00) key_value = 0;
            else if (KEY_MATRIX_STATE == 2'b01) key_value = 13;
            else if (KEY_MATRIX_STATE == 2'b10) key_value = 14;
            else if (KEY_MATRIX_STATE == 2'b11) key_value = 15;
        end

        if      (FND_STATE == 0)    seg_pos = 6'b011111;
        else if (FND_STATE == 1)    seg_pos = 6'b101111;
        else if  (FND_STATE == 2)   seg_pos = 6'b110111;
        else if  (FND_STATE == 3)   seg_pos = 6'b111011;
        else if  (FND_STATE == 4)   seg_pos = 6'b111101;
        else if  (FND_STATE == 5)   seg_pos = 6'b111110;
```

Key-matrix

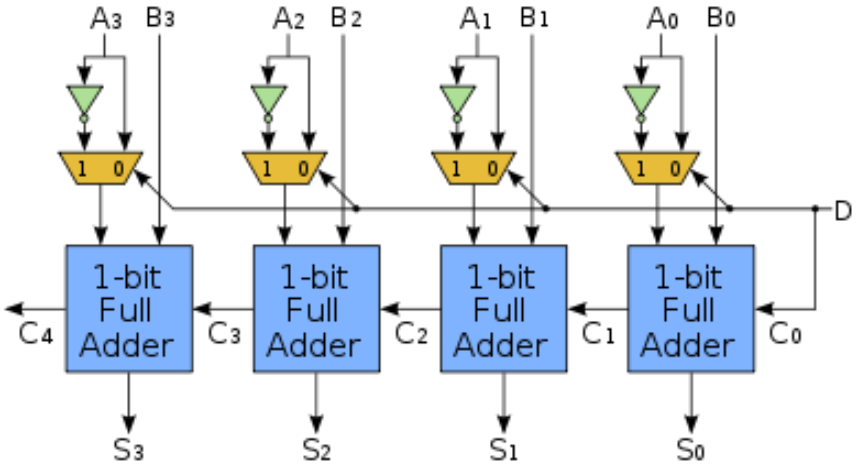
- Key-matrix 참조 코드 – 키패드를 눌러서 7-Segment에 표시하기

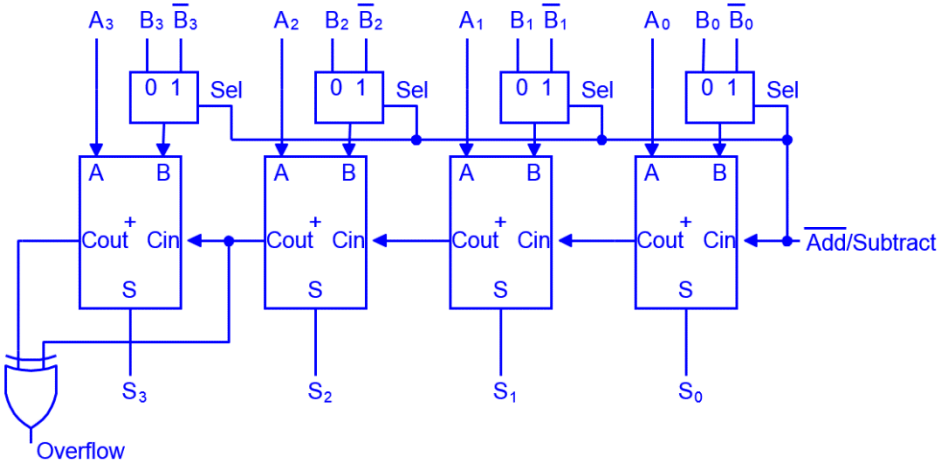
```
80          case(key_value)
81              4'b0000: seg_data = 8'b11111100;
82              4'b0001:    seg_data = 8'b01100000;
83              4'b0010:    seg_data = 8'b11011010;
84              4'b0011:    seg_data = 8'b11110010;
85
86              4'b0100:    seg_data = 8'b01100110;
87              4'b0101:    seg_data = 8'b10110110;
88              4'b0110:    seg_data = 8'b10111110;
89              4'b0111:    seg_data = 8'b11100100;
90
91              4'b1000:    seg_data = 8'b11111110;
92              4'b1001:    seg_data = 8'b11110110;
93              4'b1010:    seg_data = 8'b11101110;
94              4'b1011:    seg_data = 8'b00111110;
95
96              4'b1100:    seg_data = 8'b00011010;
97              4'b1101:    seg_data = 8'b01111010;
98              4'b1110:    seg_data = 8'b10011110;
99              4'b1111:    seg_data = 8'b10001110;
100
101             default:    seg_data = 8'b00000000;
102         endcase
103
104         if      (FND_STATE == 0)    FND_STATE = 1;
105         else if (FND_STATE == 1)    FND_STATE = 2;
106         else if  (FND_STATE == 2)   FND_STATE = 3;
107         else if  (FND_STATE == 3)   FND_STATE = 4;
108         else if  (FND_STATE == 4)   FND_STATE = 5;
109         else if  (FND_STATE == 5)   FND_STATE = 0;
110
111         if      (KEY_MATRIX_STATE == 2'b00) KEY_MATRIX_STATE = 2'b01;
112         else if (KEY_MATRIX_STATE == 2'b01) KEY_MATRIX_STATE = 2'b10;
113         else if (KEY_MATRIX_STATE == 2'b10) KEY_MATRIX_STATE = 2'b11;
114         else if (KEY_MATRIX_STATE == 2'b11) KEY_MATRIX_STATE = 2'b00;
115
116         cnt_for_device = cnt_for_device + 1;
117         if (cnt_for_device == 5121)
118             cnt_for_device = 0;
119     end
120
121 endmodule
```

Node Name	Direction	Location
clk	Input	PIN_H2
key_col[3]	Output	PIN_M11
key_col[2]	Output	PIN_M12
key_col[1]	Output	PIN_M14
key_col[0]	Output	PIN_L10
key_row[3]	Input	PIN_N15
key_row[2]	Input	PIN_N16
key_row[1]	Input	PIN_M15
key_row[0]	Input	PIN_M16
seg_data[7]	Output	PIN_A3
seg_data[6]	Output	PIN_B4
seg_data[5]	Output	PIN_B7
seg_data[4]	Output	PIN_A6
seg_data[3]	Output	PIN_A5
seg_data[2]	Output	PIN_B6
seg_data[1]	Output	PIN_B5
seg_data[0]	Output	PIN_A4
seg_pos[5]	Output	PIN_A7
seg_pos[4]	Output	PIN_B8
seg_pos[3]	Output	PIN_A8
seg_pos[2]	Output	PIN_B9
seg_pos[1]	Output	PIN_A9
seg_pos[0]	Output	PIN_A10

오늘 실습 주제: 4-bit Adder/Subtractor

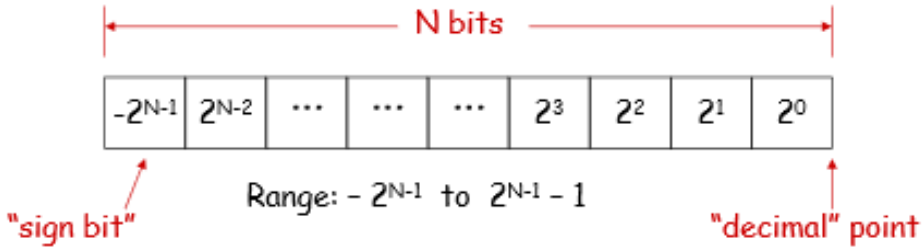
- 오늘 실습의 핵심은 제어 비트 (1bit) 하나에 의해서,
  Adder <-> Subtractor 기능 수행을 선택할 수 있는 회로를 설계하는 것에 있음
- Adder/Subtractor를 따로 만드는 것보다 효율적이고, 경제적이다.





오늘 실습 주제: 4-bit Adder/Subtractor

- 2의 보수 표기법



- 2의 보수 표기법으로 11010110는 10진수로 얼마일까?
  - $11010110 = -2^7 + 2^6 + 2^4 + 2^2 + 2^1 = -128 + 64 + 16 + 4 + 2 = -42$

- 자릿수 늘리기: 8-bit로 표시한 2의 보수를 16-bit로 표현하기

```
42 = 00101010

-5 = ~00000101 + 1
   =  11111010 + 1
   =  11111011
```

```
42 =  0000000000101010

-5 =  1111111111111011
```

MSB (= 가장 왼쪽의 bit, 여기서는 sign bit)를 그대로 늘려 쓰면 된다.
1이면 늘려진 앞의 8자리를 모두 1로, 0이면 0으로.

오늘 실습 주제: 4-bit Adder/Subtractor

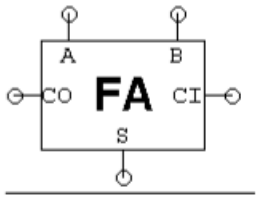
- “손”으로 푸는 2진수 덧셈의 예시:

직전 column들에서 생성된 자리 올림 수들 (carries)

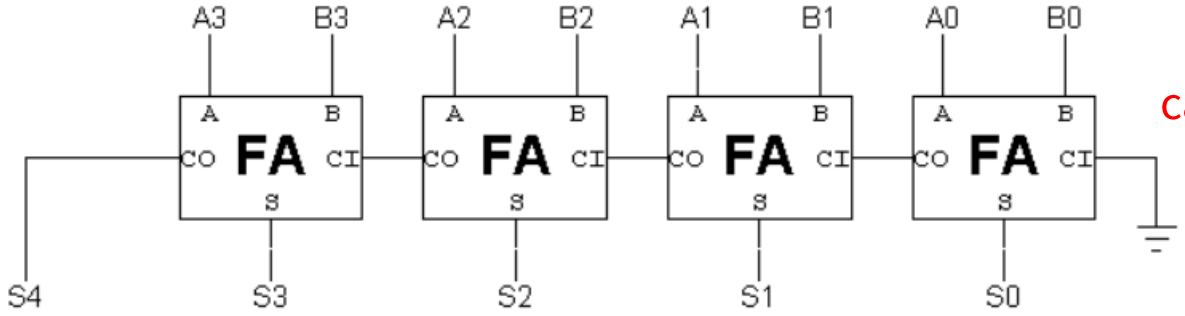
```
  1 1 0 1
    1101
+   0101
-------
   10010
```

2개의 N-bit 덧셈 연산은 최대 (N+1)-bit의 결과를 산출한다.

- FA 모듈의 인스턴스를 오른쪽과 같이 표현한다고 했을 때 :



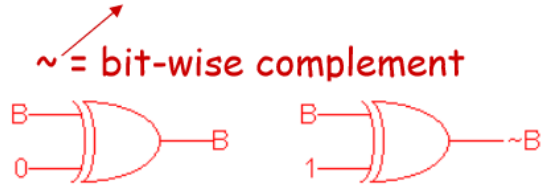
- 4-bit Adder의 경우 아래와 같이 간단히 구현할 수 있다.



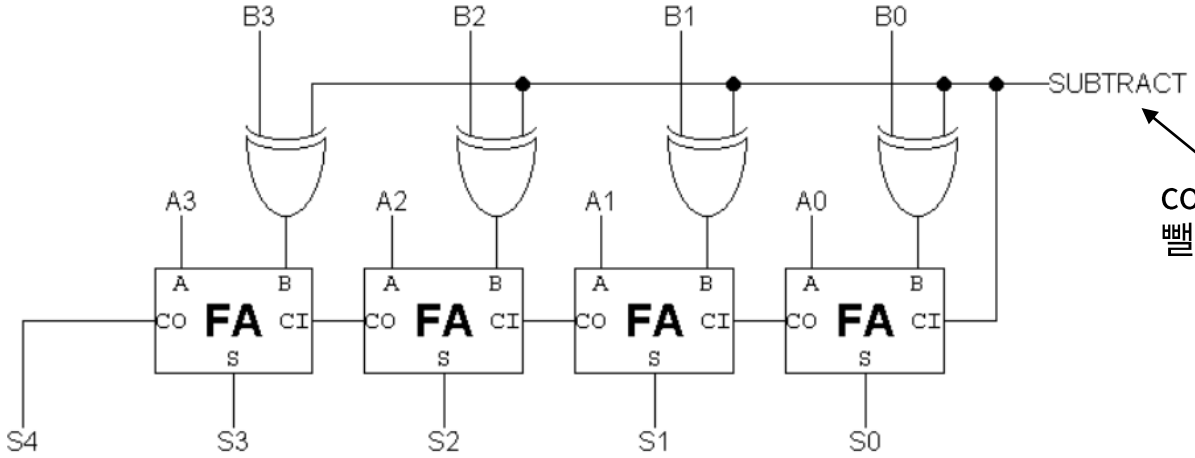
이것을 “Ripple-carry Adder”라 부른다. (마치 자리 올림수가 파도처럼 일렁인다는 의미에서)

오늘 실습 주제: 4-bit Adder/Subtractor

- Subtraction (감산): A-B = A + (-B)
- A + (-B)라 표현하는 이유 ~ (-B), 즉 B의 보수를 이용하여 감산 연산을 가산 연산처럼 수행한다는 의미에서 비롯.
- 보통 보수는 2의 보수 형태를 취한다. –B = ~B + 1



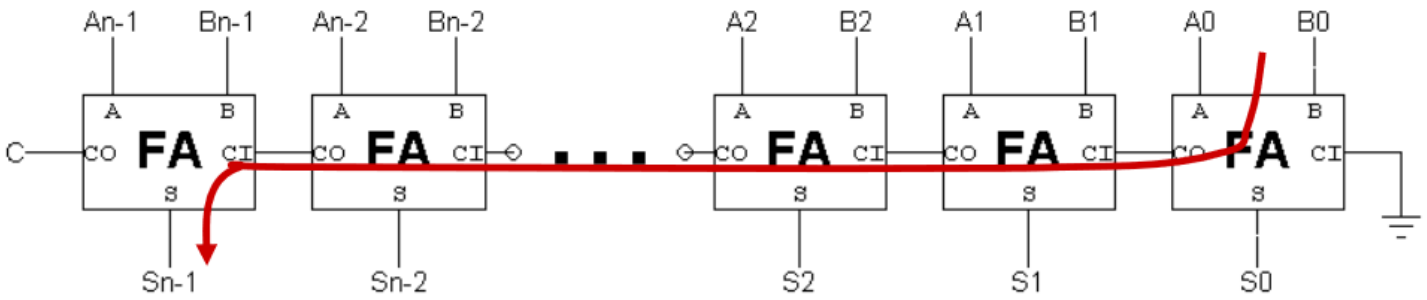
- 다음은, 오늘 만들고자 하는 '4-bit Adder/Subtractor'의 구현 물이다.



control input가 덧셈 연산이냐, 뺄셈 연산이냐를 결정한다.

[참고] 연산 속도: t_{PD} of Ripple-carry Adder

$$C_O = AB + AC_I + BC_I$$



Worst-case path (가장 느리게 결과가 나오는 경우): LSB부터 MSB까지 자리 올림수가 발생하는 경우, e.g., 11…111 + 00…001을 더할 때.

$$t_{PD} = (N-1)*(t_{PD,OR} + t_{PD,AND}) + t_{PD,XOR} \approx \Theta(N)$$

CI to CO

CI_{N-1} to S_{N-1}

$$t_{adder} = (N-1)t_{carry} + t_{sum}$$

오늘 실습 주제: 4-bit Adder/Subtractor

- [중요!] 오늘 실습에 있어서, 4-bit Adder/Subtractor를 구성하는 FA 모듈은, 다음과 같은 Gate delay의 설정 하에 진행됩니다.
- < Gate delay들 >

NOT	4.5 ns
AND	7.5 ns
OR	7.0 ns
NAND	7.5 ns
NOR	5.0 ns
XOR	6.5 ns

```
`timescale 1ns/100ps
module ... (.., .., <port list> );
       parameter  notDelay = 4.5;
       parameter  andDelay = 7.5;
       parameter   orDelay = 7.0;
       parameter nandDelay = 7.5;
       parameter  norDelay = 5.0;
       parameter  xorDelay = 6.5;
                 ......
```

- 적용 예:

```
`timescale 1ns/100ps
parameter xorDelay = 6.5;
       ......
assign #xorDelay S = A ^ B;
```

```
`timescale 1ns/100ps
parameter xorDelay = 6.5;
       ......
xor #xorDelay xor_01(S, A, B);
```