

Chapter 122. 키 매트릭스

ATmega328에는 디지털 입출력을 위한 14개의 핀과 아날로그 입력을 위한 6개의 핀이 존재하며 아날로그 입력 핀은 디지털 입출력을 위해 사용할 수 있으므로 ATmega328에서 사용할 수 있는 디지털 입출력 핀은 최대 20개이다. 20개의 입출력 핀이 많아 보일 수도 있지만 많은 수의 LED나 버튼을 사용하는 경우 20개는 그리 많지 않은 것이 사실이다. 7세그먼트 표시장치를 기억하는가? 한 자리의 숫자를 표시하기 위해서만도 8개의 핀이 필요하였고, 여러 자리의 숫자를 표시하기 위해서는 입출력 핀이 부족하여 잔상 효과를 통해 적은 수의 핀으로 많은 수의 LED를 제어할 수 있었다. 이 장에서는 적은 수의 핀으로 많은 수의 입력을 받아들일 수 있는 방법의 하나로 키 매트릭스 방법에 대해 알아본다. 키패드라고도 하는 키 매트릭스는 전화기, 키보드, 도어락 등에서 숫자 키들이 행과 열에 따라 배열된 것을 말한다. 키 매트릭스는 4×4 형태 또는 4×3 형태의 숫자 키 매트릭스를 흔히 볼 수 있으며 이 장에서는 4×4 크기의 키 매트릭스를 사용한다.



그림 1. 4×4 키 매트릭스

1. 키 매트릭스

키 매트릭스와 개념적으로 유사한 예로는 4자리 7세그먼트 표시장치와 LED 매트릭스가 있다. 4자리 7세그먼트 표시장치의 경우 특정 순간에는 하나의 자리에 해당하는 8개 LED만 켜질 수 있고 빠른 속도로 자리를 이동하면서 출력을 반복함으로써 4자리가 동시에 켜진 것과

같은 효과를 얻는다. LED 매트릭스의 경우는 특정 순간에는 하나의 열에 해당하는 8개 LED만 켜질 수 있고¹⁾ 빠른 속도로 열을 이동함으로써 8개 열이 동시에 켜진 것과 같은 효과를 얻는다. 이들은 LED를 통한 출력의 예이지만 버튼의 경우도 마찬가지이다. 특히 LED 매트릭스와 유사하게 특정 순간에는 하나의 열에 배열된 버튼의 상태를 읽어 들일 수 있으며 빠른 속도로 열을 이동하여 버튼의 상태를 읽어 들임으로써 모든 열의 버튼을 동시에 읽어 들이는 것과 같은 효과를 얻을 수 있다. 먼저 간단한 2×2 크기의 키 매트릭스를 살펴보자.

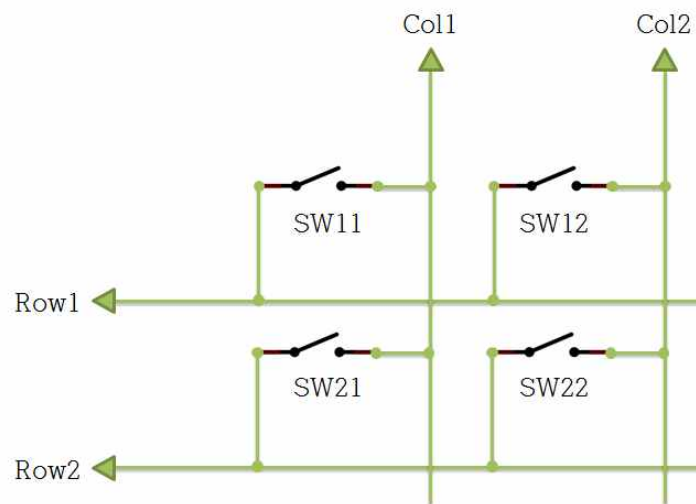


그림 2. 2×2 키 매트릭스 회로도

그림 2의 회로도에서 Col1에 HIGH가 가해지고 Col2에 LOW가 가해졌다고 가정해 보자. 1열에 연결된 버튼 SW11과 SW21이 모두 눌러지지 않았다면 Row1과 Row2는 모두 LOW 값을 가진다. 하지만 SW11이 눌러지고 SW21이 눌러지지 않았다면 Row1은 HIGH, Row2는 LOW 값을 가져 하나의 열에 있는 버튼들의 상태를 구별할 수 있다.

1) 특정 순간에 켜질 수 있는 LED는 행 또는 열 단위로 설정이 가능하며 여기서는 열 단위로 제어한다고 가정하였다.

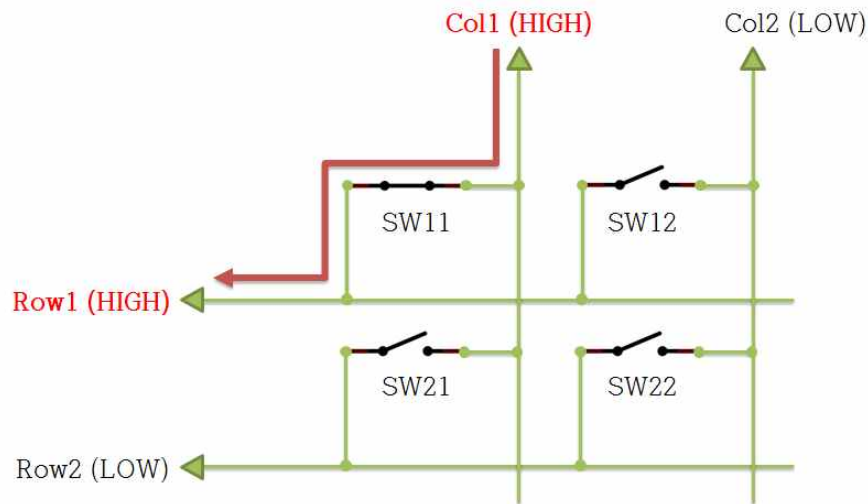


그림 3. 2×2 키 매트릭스에서 열 단위 입력 예

그림 2의 회로도에서 스위치가 눌러지지 않은 경우 각 Row에 해당하는 값이 하이 임피던스 상태에 있는 것을 막기 위해 풀다운 저항을 연결하면 키 매트릭스의 기본 회로는 완성된다.

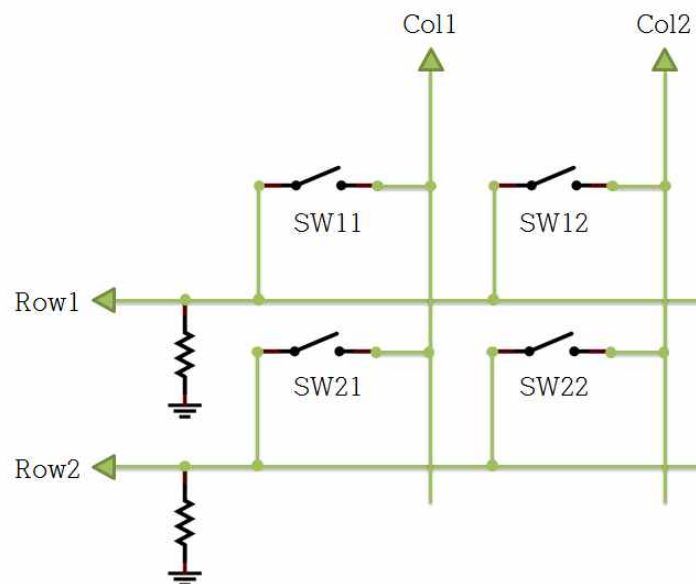


그림 4. 풀다운 저항을 연결한 2×2 키 매트릭스 회로도

그림 4의 회로도는 한 번에 하나의 버튼만이 눌러진다면 아무런 문제가 없다. 하지만 동시에 2개 이상의 버튼이 눌러지는 경우에는 잘못된 값을 얻을 수 있다. 그림 4에서 SW11과 SW12가 동시에 눌러진 경우를 생각해 보자. 첫 번째 열을 검사하기 위해 Col1에 HIGH가 Col2에 LOW가 가해진 것으로 가정한다. Row1에는 어떤 값이 가해질까?

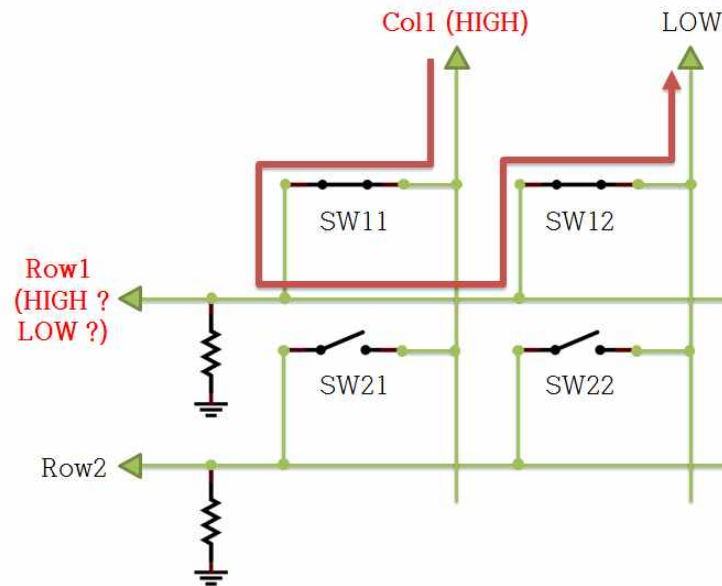


그림 5. 여러 버튼을 동시에 누름에 따른 루프 형성

그림 5에 나타난 것처럼 회로 상에서 루프가 형성되고 Row1에는 LOW 또는 HIGH 중 임의의 값이 가해질 수 있다. 또 다른 예로 SW11, SW12, SW22 3개의 버튼이 눌려졌다고 가정해 보자. SW21 버튼이 눌러지지 않았으므로 Row2에는 LOW 값이 가해질 것으로 예상되지만 그림 6에서와 같이 루프가 형성되어 HIGH 값이 가해지게 된다. 이처럼 버튼이 눌러지지 않았지만 루프 형성에 따라 버튼이 눌러진 것과 같은 값이 가해지는 현상을 고스트 (ghost) 현상 또는 고스트 키 현상이라고 한다. 고스트 현상이 발생하면 실제로 버튼이 눌러진 경우와 눌러지지 않은 경우를 구별할 수 없으며 이를 마스킹 (masking) 현상이라고 한다.

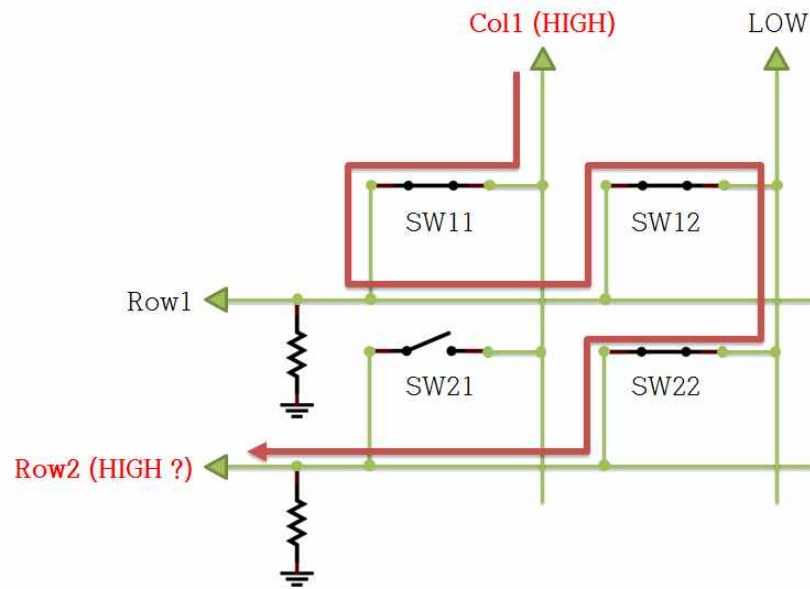


그림 6. 루프 형성에 따른 고스트 현상

고스트 현상과 고스트 현상에 의한 마스킹 현상을 없애기 위해서는 잘못된 루프가 형성되지 않도록 해주어야 하며 이를 위해서는 다이오드를 사용할 수 있다. 다이오드는 (+)에서 (-)로만 전류가 흐르도록 하는 기능을 하므로 열에서 행으로만 루프가 형성될 수 있고, 행에서 열로는 루프가 형성되지 못하도록 함으로써 고스트 현상을 억제할 수 있다.

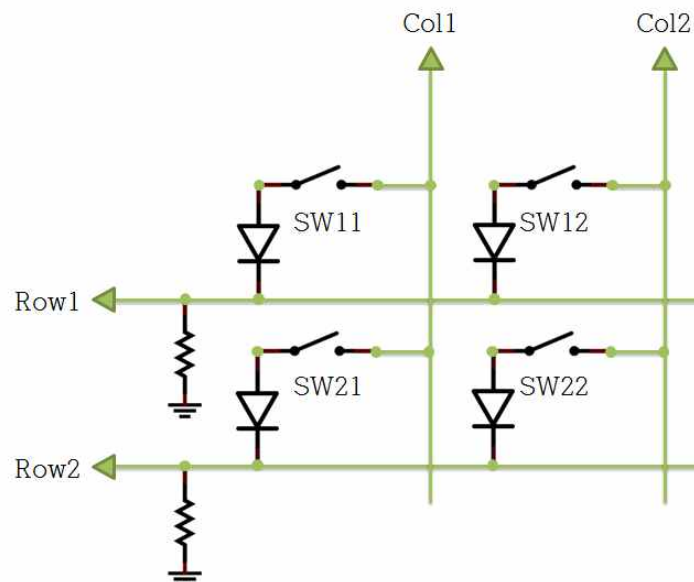


그림 7. 다이오드가 연결된 2×2 키 매트릭스 회로도

그림 7의 회로도에서 다이오드와 스위치를 LED로 교체하면 2×2 크기의 LED 매트릭스를 구현할 수 있으며 이는 이전 장에서 설명한 8×8 LED 매트릭스와 기본적으로 동일하다.

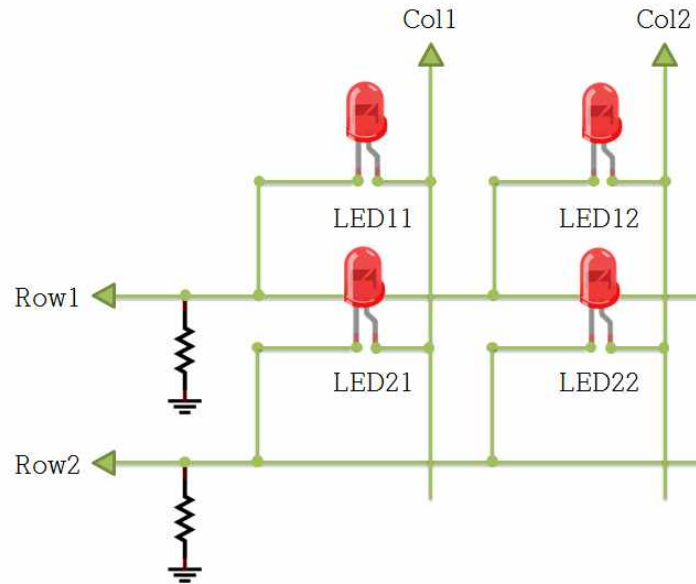


그림 8. 2×2 LED 매트릭스 회로도

그림 8의 회로에서 켜고자 하는 LED가 있는 열의 Col_n 값은 HIGH로, 행의 Row_n 값은 LOW로 설정함으로써 해당 LED 만을 켤 수 있다. 나머지 Col_n 값은 LOW로 나머지 Row_n 값은 HIGH로 설정해야 한다는 점도 잊지 말아야 한다.

2. 4×4 키 매트릭스

위에서 설명한 키 매트릭스는 2×2 크기의 키 매트릭스로 4개의 핀을 통해 4개의 버튼을 제어하는 방법을 보였다. 이는 설명을 간단하게 하기 위한 구성으로 4개의 버튼은 키 매트릭스로 구성하지 않아도 4개의 핀으로 제어가 가능하다. 키 매트릭스의 장점은 버튼의 수를 늘려보면 명백해진다. 16개의 버튼을 생각해 보자. 16개의 핀이 필요할 것으로 보이지만 4×4 키 매트릭스로 구성하면 8개의 핀만으로 제어가 가능하다. 이 장에서 사용한 그림 1의 4×4 키 매트릭스 회로도는 그림 9와 같다. 그림 9의 회로도는 그림 2의 회로도에서 행과 열의 수를 늘려놓은 것으로 그림 7에서와 같이 풀다운 저항이나 고스트 현상 방지를 위한 다이오드는 포함되어 있지 않다.

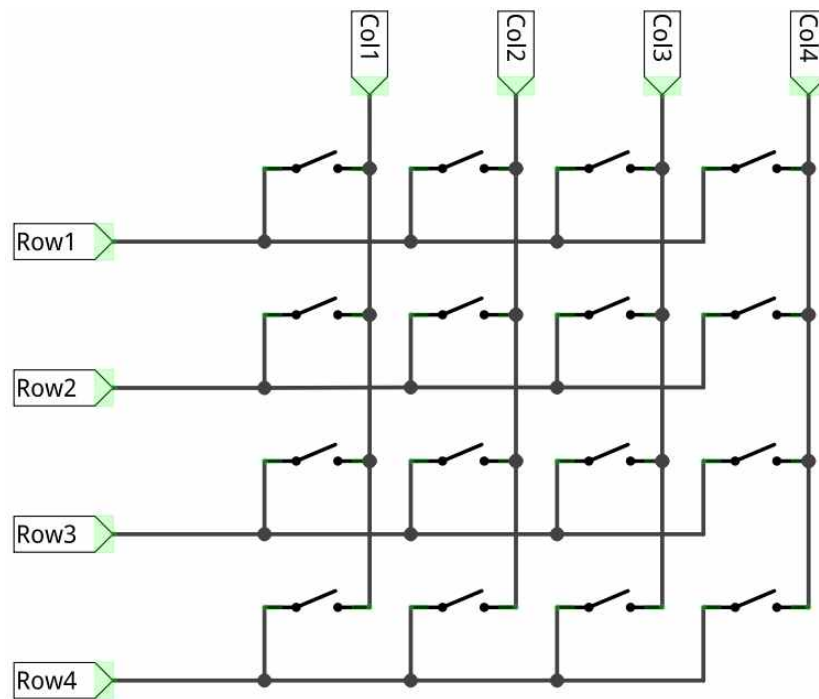


그림 9. 4×4 키 매트릭스 회로도

실험에서는 ATmega328 내부의 풀업 저항을 사용한다. 각 행을 입출력 핀과 연결할 때 내부 풀업 저항을 사용함으로써 버튼이 눌러지지 않은 경우 해당 행에는 HIGH 값이 입력되고, 버튼이 눌러진 경우 해당 행에는 LOW 값이 입력된다. 이 때 검사 대상인 열에는 LOW 값을, 검사 대상이 아닌 열에는 HIGH 값을 가해주어야 한다. 이는 위에서 설명한 경우와 반대이다.

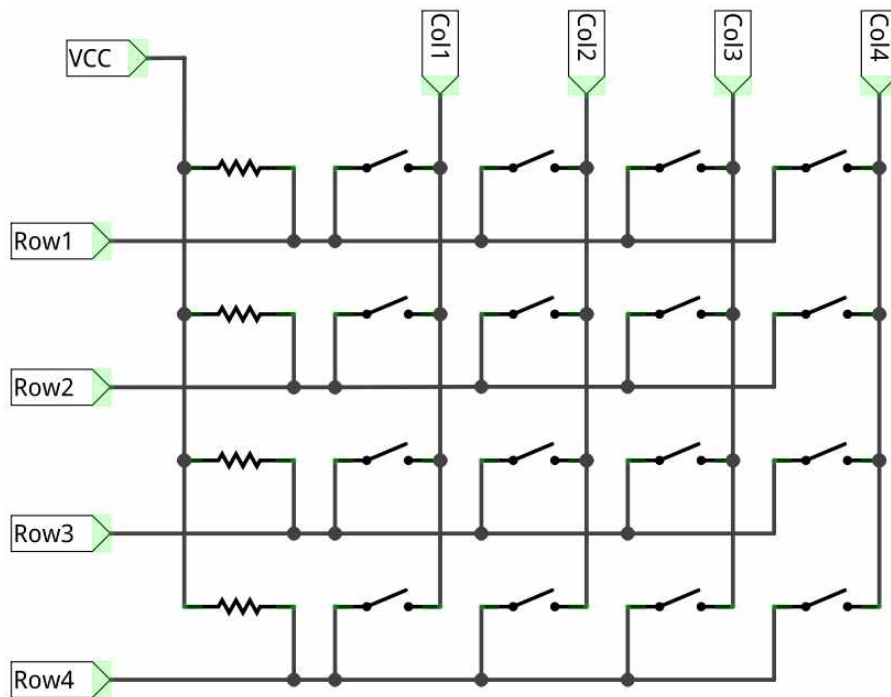


그림 10. 풀업 저항이 연결된 키 매트릭스 회로도

그림 10의 회로도에는 다이오드가 포함되어 있지 않다. 따라서 고스트 현상이 발생하며 그림 6에서와 같이 사각형을 이루는 위치의 4개 버튼 중 3개의 버튼이 눌러지면 나머지 하나의 버튼도 눌러진 것으로 인식된다. 또한 그림 5에서와 같이 동일한 열에서 2개 이상의 버튼이 눌러지면 버튼이 눌러지지 않은 것으로 인식된다. 다이오드를 사용하지 않으면 고스트 현상은 해결할 수 없지만 그림 5에서와 같은 단락 현상은 검사 대상이 아닌 열을 하이 임피던스 상태로 만들어 서로 다른 열 사이에 경로가 형성되지 않도록 해줌으로써 해결할 수 있다. 해당 열을 하이 임피던스 상태로 만들기 위해서는 해당 열이 연결된 입출력 핀을 입력 상태로 설정함으로써 가능하다. 그림 11과 같이 키 매트릭스를 ATmega328과 연결해 보자.

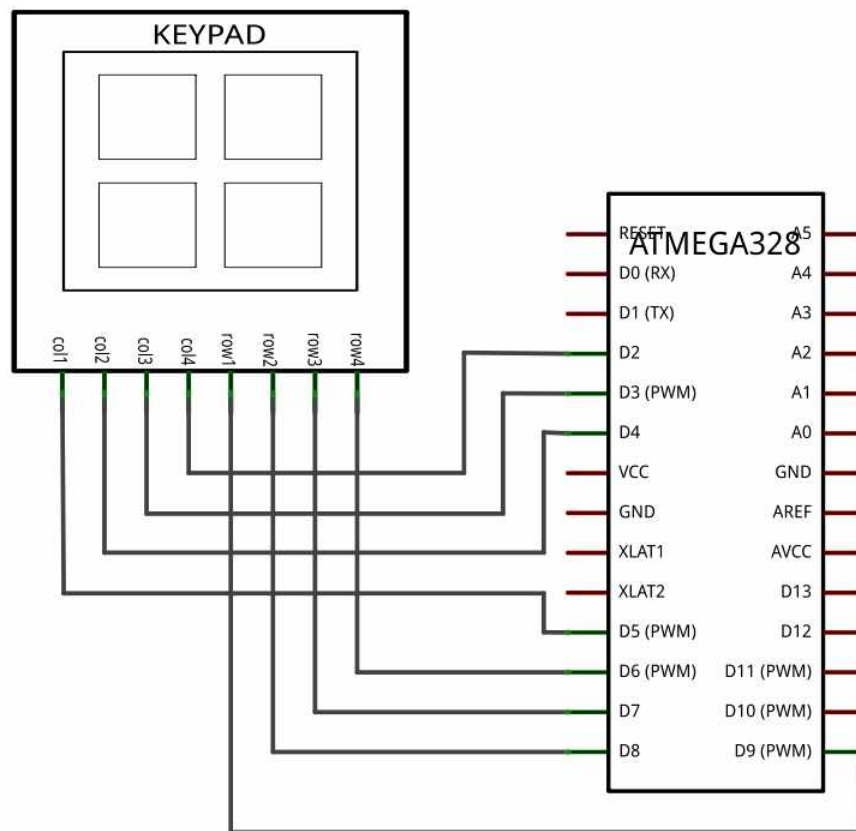


그림 11. 키 매트릭스 연결 회로도

PD0 핀(디지털 0번 핀)과 PD1 핀(디지털 1번 핀)은 버튼 상태를 UART 통신을 통해 출력하기 위해 사용하므로 PD2 핀부터 키 매트릭스 연결에 사용하였다. 코드 1은 눌러진 키를 UART 통신을 통해 출력하는 예이다. 다이오드가 사용되지 않은 키 매트릭스이므로 2개까지의 버튼은 오류 없이 감지할 수 있으나 3개 이상의 버튼이 눌러진 경우에는 고스트 현상에 의해 잘못된 결과가 나올 수 있다.

코드 1. 키 매트릭스 테스트 프로그램 - C 스타일

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include "UART.h"

#define ROWS 4
```

```
#define COLS 4

char keys[ROWS][COLS] = {           // 키 값 정의
    {'1', '2', '3', 'A'},
    {'4', '5', '6', 'B'},
    {'7', '8', '9', 'C'},
    {'*', '0', '#', 'D'}
};

uint8_t rowPins[ROWS] = {PB1, PB0, PD7, PD6};    // 행 연결 핀
uint8_t colPins[COLS] = {PD5, PD4, PD3, PD2};    // 열 연결 핀

void InitKeyMatrix(void)
{
    // 행(ROW) 연결 핀 설정 : 풀업 저항을 사용하는 입력
    DDRB &= ( ~(1 << PB1) & ~(1 << PB0) );      // 입력으로 설정
    PORTB |= ( (1 << PB1) | (1 << PB0) );        // 풀업 사용
    DDRD &= ( ~(1 << PD7) & ~(1 << PD6) );
    PORTD |= ( (1 << PD7) | (1 << PD6) );

    // 열(COL) 연결 핀 설정
    // 하이 임피던스 상태로 만들기 위해 입력으로 설정
    DDRD &= 0b11000011;
}

int main(void)
{
    uint8_t keyPressed[ROWS][COLS];             // 키 상태 저장
    uint8_t row, col, count;

    UART_INIT();                                // UART 통신 초기화
    InitKeyMatrix();                             // 키 매트릭스 연결 초기화

    while(1)
    {
        count = 0;                              // 눌려진 버튼의 개수
```

```

for(col = 0; col < COLS; col++){
    DDRD |= (1 << colPins[col]);          // 해당 COL을 출력으로 설정
    PORTD &= ~(1 << colPins[col]);        // 해당 COL에 LOW 출력
    _delay_ms(3);                          // 출력 안정화

    // LOW의 입력 읽기 : LOW인 경우 버튼이 눌려진 상태
    keyPressed[0][col] = 1 - ((PINB & (1 << rowPins[0])) >> rowPins[0]);
    keyPressed[1][col] = 1 - ((PINB & (1 << rowPins[1])) >> rowPins[1]);
    keyPressed[2][col] = 1 - ((PIND & (1 << rowPins[2])) >> rowPins[2]);
    keyPressed[3][col] = 1 - ((PIND & (1 << rowPins[3])) >> rowPins[3]);

    for(row = 0; row < ROWS; row++) // 눌려진 버튼 개수 읽기
        count += keyPressed[row][col];

    PORTD |= (1 << colPins[col]);          // 해당 COL에 HIGH 출력
    DDRD &= ~(1 << colPins[col]);          // 입력으로 (하이 임피던스 상태) 설정
}

if(count > 0){                            // 버튼이 눌려진 경우만 UART로 출력
    for(row = 0; row < ROWS; row++){
        for(col = 0; col < COLS; col++){
            if(keyPressed[row][col])
                UART_transmit(keys[row][col]);
            else
                UART_transmit('.');
            UART_printString(" ");
        }
        UART_transmit('\n');
    }
    UART_transmit('\n');
}

_delay_ms(1000);                          // 1초에 한 번만 검사
}
}

```

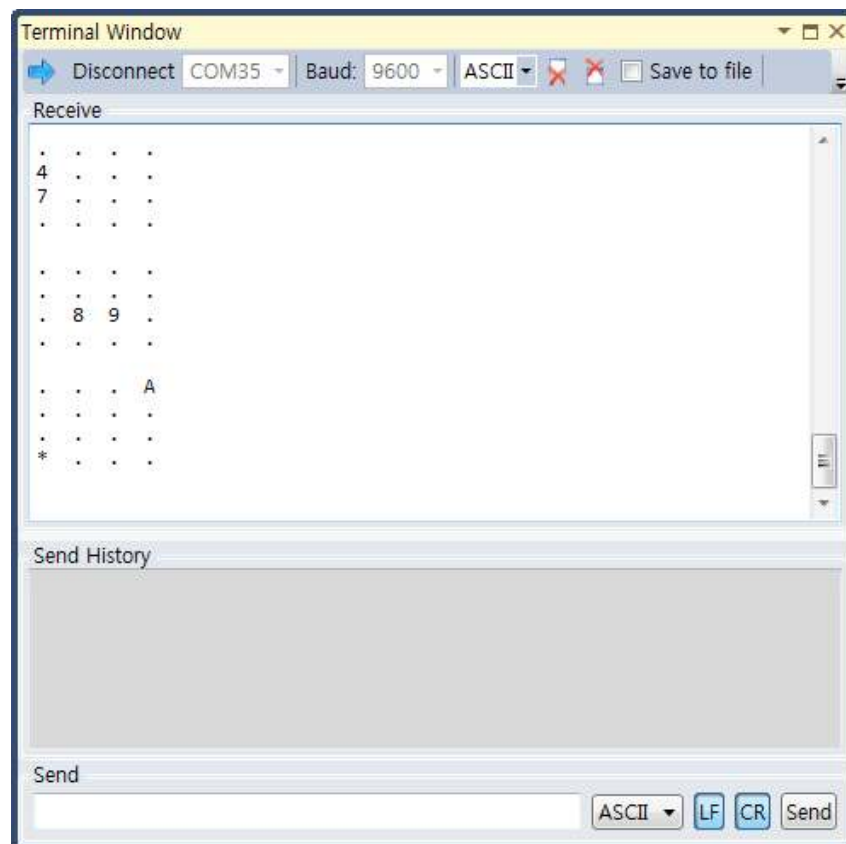


그림 12. 코드 1 실행 결과

3. 키 매트릭스 - 아두이노

아두이노에서 키 매트릭스를 사용하는 방법은 C 스타일에서와 다르지 않다. 코드 2는 코드 1을 아두이노 환경에서 사용할 수 있도록 수정한 예이다. C 스타일에서는 8비트 포트 단위의 데이터 입출력을 기본으로 하므로 비트 연산을 통한 레지스터 조작이 필요하다. 하지만 아두이노에서는 비트 단위의 데이터 입출력을 기본으로 하며 각 핀에 번호가 할당되어 있으므로 코드를 읽고 이해하기가 쉽다. 코드 2의 실행 결과는 코드 1의 실행 결과와 동일하다.

코드 2. 키 매트릭스 테스트 프로그램 - 아두이노 스타일

```
const byte ROWS = 4;
const byte COLS = 4;

char keys[ROWS][COLS] = {           // 키 값 정의
```

```
{'1', '2', '3', 'A'},
{'4', '5', '6', 'B'},
{'7', '8', '9', 'C'},
{'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};           // 행 연결 핀
byte colPins[COLS] = {5, 4, 3, 2};          // 열 연결 핀
byte pattern[ROWS][COLS];                   // 키 상태 저장

void setup(){
    // 행 연결 핀을 풀업 저항을 사용한 입력으로 설정
    for(int i = 0; i < ROWS; i++)
        pinMode(rowPins[i], INPUT_PULLUP);

    // 열 연결 핀은 하이 임피던스 상태로 만들기 위해 입력으로 설정
    for(int i = 0; i < COLS; i++)
        pinMode(colPins[i], INPUT);

    Serial.begin(9600);                      // 직렬 통신 초기화
}

void loop(){
    int row, col, count = 0;

    for(col = 0; col < COLS; col++){
        pinMode(colPins[col], OUTPUT);       // 해당 COL을 출력으로 설정
        digitalWrite(colPins[col], LOW);     // 해당 COL에 LOW 출력

        for(row = 0; row < ROWS; row++){    // LOW의 입력 읽기
            pattern[row][col] = !digitalRead(rowPins[row]);
            count += pattern[row][col];
        }
        digitalWrite(colPins[col], HIGH);    // 해당 COL에 HIGH 출력
        pinMode(colPins[col], INPUT);        // 입력으로 (하이 임피던스 상태) 설정
    }
}
```

```

if(count > 0){                                     // 버튼이 눌러진 경우만 시리얼로 출력
    for(row = 0; row < ROWS; row++){
        for(col = 0; col < COLS; col++){
            if(pattern[row][col])
                Serial.write(keys[row][col]);
            else
                Serial.write('.');
            Serial.write(" ");
        }
        Serial.write('\n');
    }
    Serial.write('\n');

}

delay(1000);
}

```

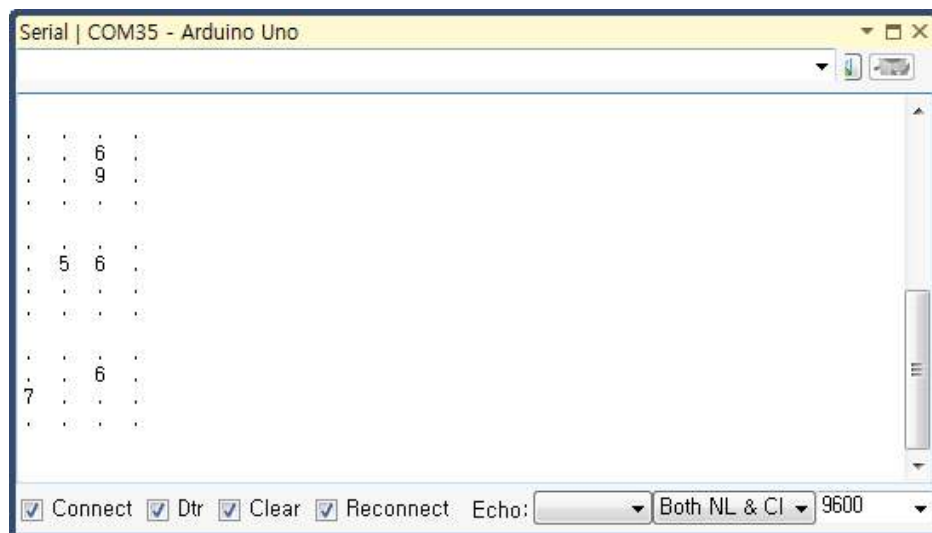


그림 13. 코드 2 실행 결과

아두이노에서 키 매트릭스를 사용하기 위해서는 공개된 전용의 키패드 라이브러리를 사용할 수 있다. 먼저 키패드 라이브러리를 다운 받아 설치한다.²⁾ 카패드 라이브러리는 기본적으로

여러 개의 키가 눌려진 상태를 검사할 수 있으며 이외에도 다양한 유틸리티 함수를 제공하고 있으므로 간편하게 사용할 수 있다. 먼저 동시에 두 개 이상의 키가 눌러지지 않는다고 가정하고 현재 눌려진 키를 검사하는 프로그램을 키패드 라이브러리를 사용하여 만들어 보자. 코드 3은 키패드 라이브러리를 사용하여 현재 눌려진 버튼을 출력하는 예이다.

코드 3. 키패드 라이브러리

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

// 키패드 객체 생성 및 초기화
// 키패드의 크기, 연결 핀 그리고 키 값을 지정
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

void setup(){
  Serial.begin(9600);           // 직렬 통신 초기화
}

void loop(){
  char key = keypad.getKey();    // 현재 눌려진 키 검사
  if(key != NO_KEY){            // 눌려진 키가 있는 경우
    Serial.println(String("Currently pressed key is ") + key);
  }
}
```

2) <http://playground.arduino.cc/Code/Keypad>

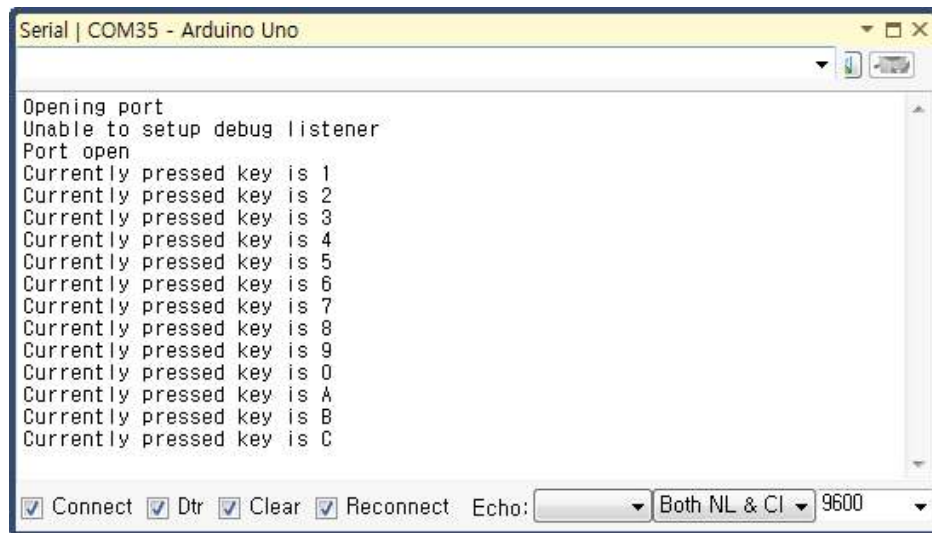


그림 14. 코드 3 실행 결과

키패드 라이브러리의 getKey 함수는 char 형의 키 값을 반환하며, 두 개 이상의 키가 눌려진 경우에는 첫 번째 키 값을 반환한다. 두 개 이상의 키가 동시에 눌려진 경우를 검사하고 싶다면 getKey 함수를 사용하면 된다. 보다 자세한 내용은 키패드 라이브러리의 예제를 참고하기 바란다.