

11. 디지털 입출력 확장 - MCP23017

이전 장에서 74595 직렬 입력 병렬 출력 레지스터 칩을 이용하여 3개의 아두이노 핀으로 8개의 디지털 출력을 사용하는 방법을 살펴보았다. 74595 칩은 칩을 직렬로 연결함으로써 3개의 아두이노 핀으로 16개 이상의 출력도 사용할 수 있도록 해주므로 여러 개의 LED를 제어하기 위해 흔히 사용된다. 하지만 74595 칩은 디지털 출력만을 확장해줄 수 있는 한계가 있다. 이 장에서는 MCP23017 칩을 이용하여 16개의 디지털 입출력을 확장할 수 있는 방법에 대해 알아본다. MCP23017 칩은 I2C 프로토콜을 사용하며 8개의 칩을 연결할 수 있으므로 최대 128개의 입출력을 확장할 수 있으며 아두이노의 두 개의 핀만을 사용한다. 다만 아두이노 우노에서는 I2C를 위한 전용 핀인 아날로그 A4 및 A5 핀만을 사용할 수 있다.

1. I2C (Inter-Integrated Circuit)

I2C는 마이크로컨트롤러와 저속의 주변 장치 사이에 디지털 정보를 간편하게 전송할 수 있는 방법을 제공하기 위해 필립스에서 개발한 직렬 통신 프로토콜 중 하나이다. I2C 통신을 위해서는 SCL(serial clock)과 SDA(serial data) 두 개의 연결만이 필요하며 아두이노 소프트웨어에는 I2C 통신을 지원하기 위한 Wire 라이브러리가 포함되어 있다. 또한 아두이노 우노 보드는 SCL을 위해 아날로그 5번 핀을, SDA를 위해 아날로그 4번 핀을 사용하도록 지정되어 있다.

I2C의 장점은 신호선 2개만 연결하면 사용할 수 있으며 여러 개의 장치를 연결하는 경우에도 필요한 신호선의 수가 증가하지 않는다는 점이다. 하지만 I2C는 다른 프로토콜에 비해 속도가 느리다는 단점이 있다. 또한 I2C는 반이중(half-duplex) 방식으로 송신과 수신에 동시에 이루어질 수 없기 때문에 양방향 통신이 필요한 경우에는 전송 속도가 더 느려진다. 따라서 아두이노에서는 일반적으로 많은 데이터 전송을 요구하지 않는 센서를 위해 I2C 통신을 주로 사용한다.

I2C 통신은 마스터-슬레이브 (master-slave) 구조로 이루어지며 연결의 한 쪽에는 유일한 마스터(master) 장치가 존재하여 다른 슬레이브(slave) 장치들과의 정보 전송을 책임진다. 일반적으로 아두이노가 마스터의 역할을 수행하며 아두이노와 연결된 하나 이상의 장치들이 슬레이브 역할을 한다. 여러 개의 슬레이브 장치들이 두 개의 동일한 데이터 선을 공유하므로 슬레이브 장치를 구별하기 위해 I2C에서는 슬레이브의 주소를 사용한다. 슬레이브 장치들은 7비트의 고유 주소에 의해 식별되므로 최대 $2^7 = 128$ 개의 슬레이브 장치가 연결될 수 있다. 그림 1

은 I2C 버스를 통해 여러 개의 슬레이브 장치들이 연결된 예를 보여준다.

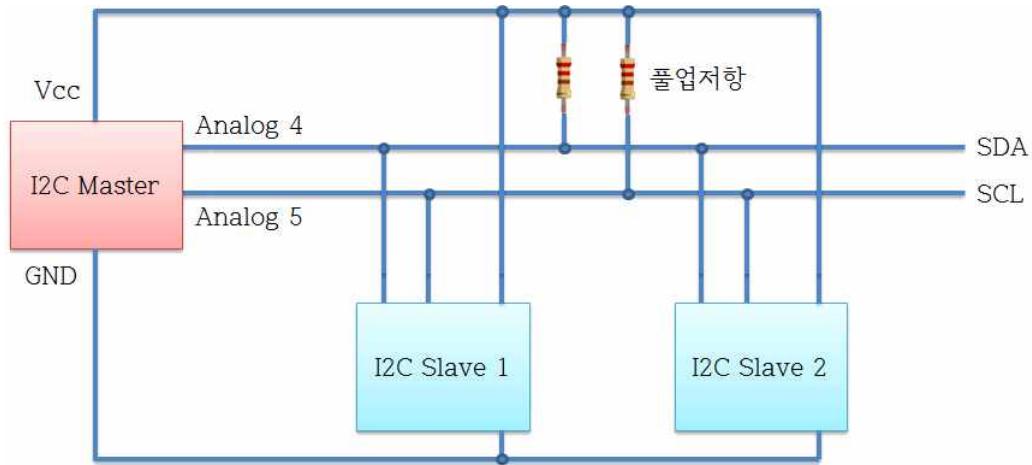


그림 1. I2C 연결

2. MCP23017

MCP23017은 16비트 디지털 입출력 확장 칩으로 16개의 디지털 입출력을 I2C 프로토콜을 이용하여 추가적으로 이용할 수 있도록 해준다. MCP23017은 최대 8개까지 동시에 연결하여 사용할 수 있으므로 최대 128개의 추가적인 입출력 포트를 제공할 수 있다. MCP23017은 그림 28 핀으로 이루어진 칩으로 16비트의 입출력을 8비트 단위의 두 그룹(bank)으로 나누어 관리한다.

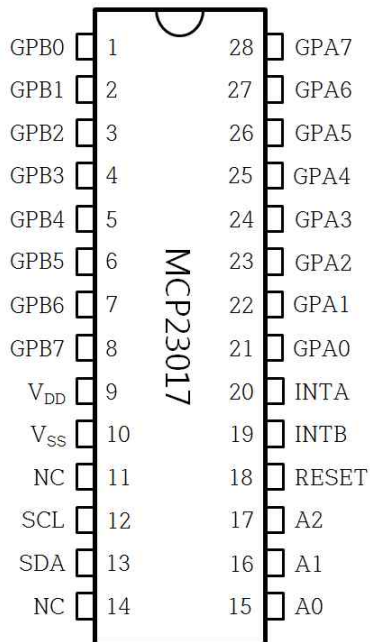


그림 2. MCP23017 칩 핀 배치도

그림 2에서 9번 핀 VDD는 5V에, 10번 핀 VSS는 GND에 연결하는 전원 핀이다. 11번과 14번 핀은 사용하지 않으며 (NC, Not Connected), 12번과 13번 각각 I2C 연결을 위한 클록과 데이터 선으로 아두이노 우노와는 SCL을 아날로그 5번 핀에 SDA를 아날로그 4번 핀에 연결한다. I2C 연결에서는 안정적인 데이터 전송을 위해 외부 풀업저항 연결이 필요하며 일반적으로 10kΩ 저항을 풀업저항으로 사용한다. 18번 RESET 핀은 GND에 연결하여 칩을 초기화하기 위해 사용하는 핀으로 (Active Low) 5V에 연결하여 아무런 동작을 수행하지 않도록 한다. 19번과 20번 핀은 인터럽트 관련 핀으로 이 장에서는 사용하지 않는다. 마지막 15번에서 17번까지의 핀은 I2C 연결을 위한 주소 시정을 위해 사용되는 핀이다. MCP23017 칩의 기본 I2C 주소는 $0x20 = 0010\ 0000_2$ 으로 여기에 $(A_2A_1A_0)_2$ 값을 더해 주소가 결정된다. 만약 A_2 를 5V에 연결하고 A_1 과 A_0 를 GND에 연결하였다면 MCP23017 칩의 I2C 주소는 $0010\ 0100_2 = 0x24$ 가 될 것이다. 이처럼 MCP23017 칩에는 0x20에서 0x27까지 8개의 서로 다른 주소를 지정할 수 있으므로 $16 \times 8 = 128$ 개의 입출력을 I2C를 이용하여 확장할 수 있다. DM-UStyle EX 보드에 포함되어 있는 MCP23017 칩의 I2C 주소는 0x20으로 설정되어 있으며 변경이 불가능하다.

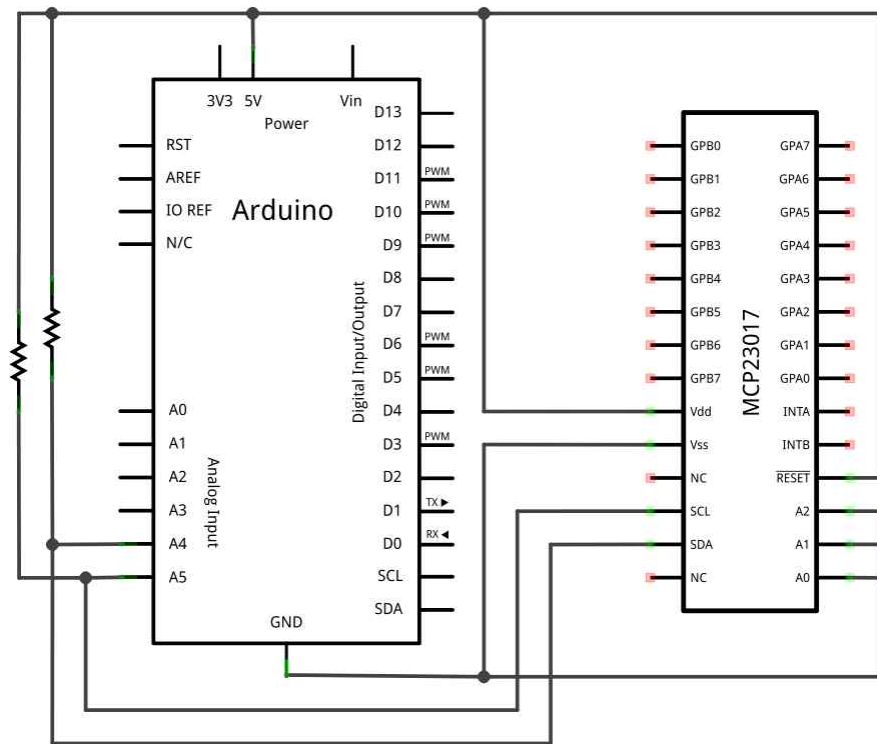


그림 3. MCP23017 연결 회로도

먼저 MCP23017 칩의 뱅크 B에 8개의 LED를 연결해 보자. DM-UStyle EX 보드에는 뱅크 B에 연결된 LED가 포함되어 있다.

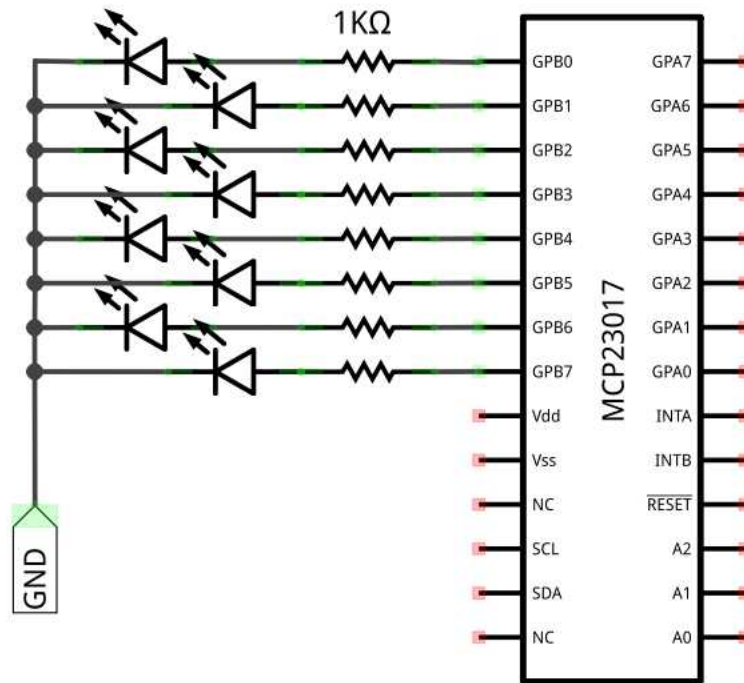


그림 4. MCP23017에 LED 연결 회로도

MCP23017 칩은 I2C 통신을 사용하며 전용 라이브러리가 공개되어 있다. 먼저 MCP23017 칩을 제어하기 위한 라이브러리를 다운받아¹⁾ 스케치북 디렉터리의 'libraries' 디렉터리 아래에 설치한다. 라이브러리에서는 16개의 확장된 디지털 입출력 핀을 0에서 15번까지의 번호로 관리하고 있다. 이 때 0번은 GPA0(MCP23017의 21번 핀)에 해당하고 15번은 GPB7(MCP23017의 8번 핀)에 해당한다. 라이브러리를 사용하기 위해서는 먼저 해당 헤더 파일을 포함시켜야 한다. MCP23017 칩은 I2C 프로토콜을 사용하므로 Wire 라이브러리의 헤더 파일 역시 포함시켜야 한다. 아두이노 프로그램에서 '스케치 → 라이브러리 가져오기... → Wire'와 '스케치 → 라이브러리 가져오기... → Adafruit_MCP23017' 메뉴를 선택하여 해당 헤더 파일을 포함시킨다.

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>
```

헤더 파일이 포함되면 Adafruit_MCP23017 클래스의 객체를 생성한다. MCP23017은 동시에

1) <https://github.com/adafruit/Adafruit-MCP23017-Arduino-Library>

8개까지 연결할 수 있으므로 여러 개의 칩을 동시에 사용하는 경우에는 칩별로 하나씩의 객체를 생성하여야 한다.

```
Adafruit_MCP23017 mcp;
```

생성된 객체는 begin 함수를 통해 초기화를 수행하여야 하며 초기화에서는 MCP23017 칩의 주소를 설정한다. 주소는 0에서 7까지의 값이 지정될 수 있으며 디폴트값은 0으로 0x20을 기본 I2C 주소로 사용한다. DM-UStyle EX 보드의 MCP23017 칩은 0x20의 기본 주소를 사용한다.

```
void begin(uint8_t addr)      // I2C 주소로 (0x20 + addr)을 사용
void begin(void)              // I2C 주소로 0x20을 사용
```

주소가 지정되면 사용하고자 하는 핀의 모드를 지정한다. MCP23017 칩은 입력(INPUT) 또는 출력(OUTPUT)으로 사용할 수 있으므로 모드 지정이 반드시 필요하다.

```
mcp.pinMode(8, OUTPUT);      // 0에서 15번까지 번호를 지정할 수 있다.
```

모드가 지정되면 해당 핀으로의 출력은 digitalWrite 함수를 통해 아두이노의 디지털 출력 핀과 동일한 방식으로 제어가 가능하다. 스케치 1은 MCP23017 칩의 बैं크 B에 연결된 8개의 LED를 점멸하는 스케치의 예이다.

스케치 1. MCP23017 칩의 बैं크 B 출력 제어

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>

Adafruit_MCP23017 mcp;      // MCP23017 칩 객체
```

```
void setup() {
  mcp.begin();                // 기본 0x20 주소 사용

  for(int i = 8; i < 16; i++){
    mcp.pinMode(i, OUTPUT);    // बैं크 B의 핀을 출력으로 사용
  }
}

void loop() {
  for(int i = 0; i < 2; i++){
    for(int j = 8; j < 16; j++){
      if(j % 2 == i)
        mcp.digitalWrite(j, HIGH);
      else
        mcp.digitalWrite(j, LOW);
    }
    delay(200);
  }
}
```

MCP23017 칩의 बैं크 A에 8개의 버튼을 연결해 보자. MCP23017 칩에는 풀업저항이 포함되어 있으므로 버튼만을 연결하면 된다.

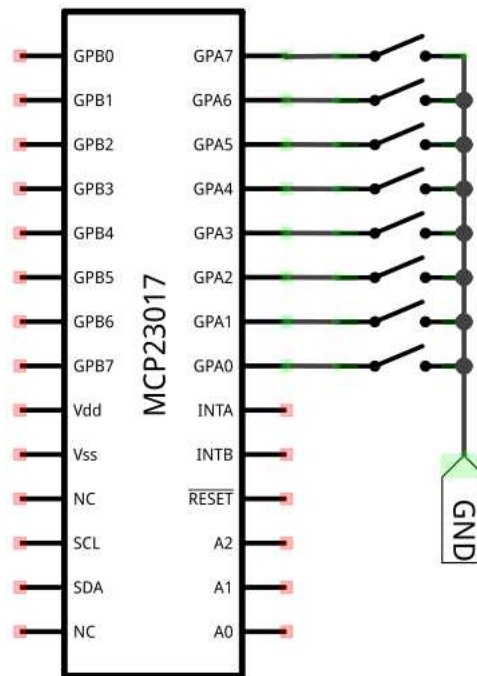


그림 5. MCP23017에 버튼 연결 회로도

먼저 해당 핀의 상태를 입력으로 설정한다.

```
mcp.pinMode(0, INPUT);           // 0에서 15번까지 번호를 지정할 수 있다.
```

입력으로 사용하는 경우에는 MCP23017 칩의 내장 풀업저항을 사용할 수 있으므로 풀업저항을 사용하도록 설정한다.

```
mcp.pullUp(0, HIGH);             // 풀업저항을 사용하도록 설정
```

설정이 끝난 핀으로부터의 입력은 digitalRead 함수를 통해 아두이노의 디지털 입력 핀과 동일한 방식으로 제어가 가능하다. 스케치 2은 MCP23017 칩의 뱅크 A에 연결된 8개의 버튼 상태를 읽어 시리얼 모니터로 출력하는 스케치의 예이다.

스케치 2. MCP23017 칩의 뱅크 A 입력 제어

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>

Adafruit_MCP23017 mcp;          // MCP23017 칩 객체

void setup() {
  mcp.begin();                  // 기본 0x20 주소 사용

  for(int i = 0; i < 8; i++){
    mcp.pinMode(i, INPUT);      // 버튼이 연결된 핀을 입력으로 설정
    mcp.pullUp(i, HIGH);        // 풀업저항을 사용하도록 설정
  }
  Serial.begin(9600);
}

void loop() {
  for(int i = 0; i < 8; i++){
    boolean buttonState = mcp.digitalRead(i);    // 버튼 상태 읽기

    // 풀업 저항을 사용하였으므로 누르지 않은 상태에서 HIGH가 입력된다.
    if(buttonState){
      Serial.print("X ");
    }
    else{
      Serial.print("O ");
    }
  }
  Serial.println();             // 줄 바꾸기

  delay(1000);
}
```

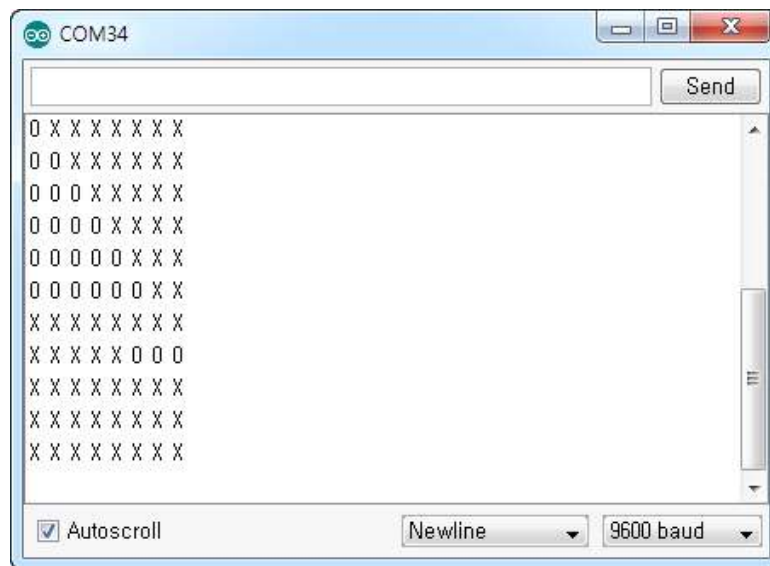


그림 6. 스케치 2 실행 결과

스케치 2에서는 내장 풀업저항을 사용하였으므로 버튼이 눌러지지 않은 경우 HIGH 입력이, 버튼이 눌러진 경우 LOW 입력이 주어지는 점에 유의하여야 한다.

3. 미니 피아노

MCP23017칩의 뱅크 A에는 8개 버튼을, 뱅크 B에는 8개의 LED가 연결해 보자. 먼저 버튼을 누르는 경우 해당 LED가 켜지도록 스케치를 작성해 보자. 버튼과 LED는 GPAn (n = 0, ... , 7)에 연결된 버튼은 GPBn에 연결된 LED로 연결되도록 한다.

스케치 3. 미니 피아노 I

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>

Adafruit_MCP23017 mcp;          // MCP23017 칩 객체

void setup() {
  mcp.begin();                  // 기본 0x20 주소 사용

  for(int i = 0; i < 8; i++){
```

```

    mcp.pinMode(i, INPUT);      // 버튼이 연결된 핀을 입력으로 설정
    mcp.pullUp(i, HIGH);        // 풀업저항을 사용하도록 설정
}
for(int i = 8; i < 16; i++){
    mcp.pinMode(i, OUTPUT);     // LED 연결 핀을 출력으로 설정
}
}

void loop() {
    for(int i = 0; i < 8; i++){
        boolean buttonState = mcp.digitalRead(i);
        // 풀업 저항을 사용하였으므로 LED로는 반전시켜 출력하여야 한다.
        mcp.digitalWrite(i + 8, !buttonState);
    }
}

```

버튼을 누를 경우 해당 LED에 불이 켜지는 것과 함께 스피커로 음이 재생되도록 해 보자. 먼저 스피커를 디지털 4번 핀에 연결한다.

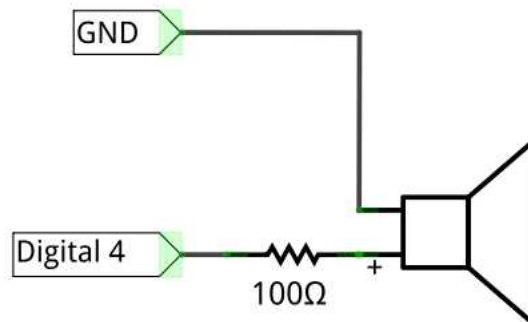


그림 7. 스피커 연결 회로도

음 재생을 위해서는 `pitches.h`²⁾ 파일을 포함시켜야 한다. 음 재생 시간은 버튼이 한 번 눌려지는 경우 100ms로 하자. 스케치 4는 스케치 3을 바탕으로 LED로 불이 켜지면서 음이 재생되도록 수정한 스케치이다.

2) `pitches.h` 파일은 디지털 데이터 출력 장에서 라이브러리로 설치하였다. `pitches.h` 파일 설치에 대한 내용은 해당 장을 참고하기 바란다.

스케치 4. 미니 피아노 II

```
#include <Wire.h>
#include <Adafruit_MCP23017.h>
#include <pitches.h>           // 음 높이 정의

Adafruit_MCP23017 mcp;        // MCP23017 칩 객체

int speaker_pin = 4;
int duration = 100;

int notes[] = {  NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4,
                 NOTE_G4, NOTE_A4, NOTE_B4, NOTE_C5};

void setup() {
  pinMode(speaker_pin, OUTPUT);    // 스피커 연결 핀을 출력으로 설정

  mcp.begin();                     // 기본 0x20 주소 사용

  for(int i = 0; i < 8; i++){
    mcp.pinMode(i, INPUT);          // 버튼이 연결된 핀을 입력으로 설정
    mcp.pullUp(i, HIGH);            // 풀업저항을 사용하도록 설정
  }
  for(int i = 8; i < 16; i++){
    mcp.pinMode(i, OUTPUT);         // LED 연결 핀을 출력으로 설정
  }
}

void loop() {
  for(int i = 0; i < 8; i++){
    boolean buttonState = mcp.digitalRead(i);
    // 풀업 저항을 사용하였으므로 LED로는 반전시켜 출력하여야 한다.
    if(!buttonState){
      mcp.digitalWrite(i + 8, !buttonState);    // 해당 LED 켜기
      tone(speaker_pin, notes[i], duration);    // 해당 음 재생
    }
  }
}
```

```
    else{  
        mcp.digitalWrite(i + 8, !buttonState);  
    }  
}  
}
```

재생 가능한 음이 8개뿐이지만 버튼을 눌러서 학교종이 땡땡땡 정도는 연주할 수 있다.

4. 4자리 7세그먼트 모듈

4자리 7세그먼트 모듈은 각 세그먼트 제어를 위해 8개, 각 자릿수 선택을 위해 4개, 총 12개의 연결을 필요로 한다. 이처럼 많은 연결을 필요로 하는 경우에는 MCP23017을 사용하여 2개의 아두이노 핀만으로 제어할 수 있다. 먼저 4자리 7세그먼트 모듈을 MCP23017의 출력과 연결해 보자. 뱅크 A에는 각 세그먼트 제어 핀을 연결하고, 뱅크 B의 0에서 3번 출력을 자릿수 선택 핀을 연결해 보자.

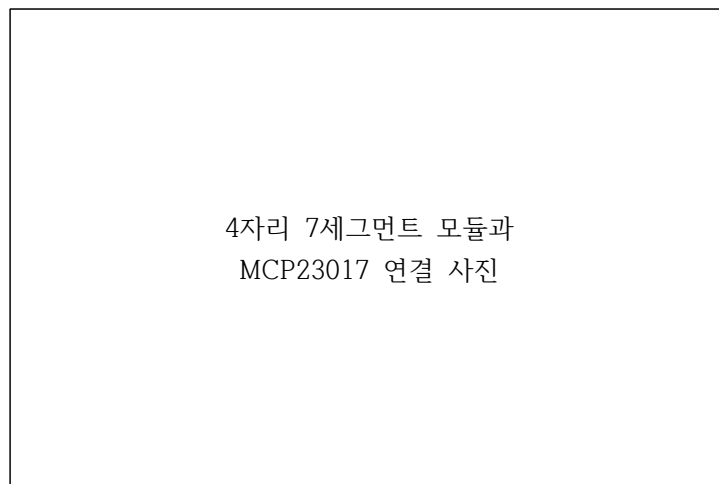


그림 8. 4자리 7세그먼트 모듈과 MCP23017 연결

해당 위치에 해당 숫자를 표시할 수 있는 show_digit 함수는 다음과 같다.

```

void show_digit(int pos, int number){    // (위치, 출력할 숫자)
    // 상위 4비트는 사용되지 않는다.
    int position_data = 0b00001111;
    // 해당 자리의 제어 선에만 LOW 값을 가하므로 해당 비트를 0으로 만든다.
    bitClear(position_data, (pos - 1));

    // 숫자 데이터. 세그먼트 a부터 세그먼트 dp 순서
    int number_data = patterns[number];

    // 자릿수 데이터와 숫자 데이터로 16비트 데이터 조합
    int output_data = (position_data << 8) | number_data;

    // MCP23017로 출력
    mcp.writeGPIOAB(output_data);
}

```

MCP23017칩의 각 입출력 핀으로는 비트 단위의 데이터를 출력할 수 있지만 16비트 전체를 한꺼번에 읽거나 쓸 수 있는 함수 역시 제공하고 있다. `wirteGPIOAB` 함수는 $GPB_7 \dots GPB_0$ $GPA_7 \dots GPA_0$ 순으로 배열된 16비트 값을 한 번에 출력하도록 해준다. 이 중 GPB7에서 GPB4까지는 실제로 사용되지 않으며 GPB3에서 GPB0까지만 자릿수 선택을 위해 사용된다. 4자리 7세그먼트 모듈은 공통 음극 방식으로 자릿수 선택을 위해서는 LOW 입력을 가해주어야 하므로 `bitClear` 함수를 통해 해당 위치의 비트만을 영(zero)로 설정하였다. 숫자 데이터는 세그먼트 a부터 세그먼트 dp 순서로 정의되어 있으므로 모듈 연결 시에 순서에 주의하도록 한다. 스케치 5는 MCP23017 칩에 연결된 4자리 7세그먼트 모듈이 정상적으로 동작하는지를 테스트하기 위한 코드로 각 자릿수에 0부터 9까지 숫자를 출력하기를 반복한다.

스케치 5. MCP23017에 연결된 4자리 7세그먼트 모듈 제어

```

#include <Wire.h>
#include <Adafruit_MCP23017.h>

// 0에서 9까지 숫자 표현을 위한
// 세그먼트 a, b, c, d, e, f, g, dp의 패턴

```

```
byte patterns[] = {
    0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE4, 0xFE, 0xE6
};

Adafruit_MCP23017 mcp;          // MCP23017 칩 객체

void setup() {
    mcp.begin();                 // 기본 0x20 주소 사용

    for(int i = 0; i < 16; i++){
        mcp.pinMode(i, OUTPUT); // 버튼이 연결된 핀을 입력으로 설정
    }
}

// 해당 자리에 숫자 하나를 표시하는 함수
void show_digit(int pos, int number){ // (위치, 출력할 숫자)
    // 상위 4비트는 사용되지 않는다.
    int position_data = 0b00001111;
    // 해당 자리의 제어 선에만 LOW 값을 가하므로 해당 비트를 0으로 만든다.
    bitClear(position_data, (pos - 1));

    // 숫자 데이터. 세그먼트 a부터 세그먼트 dp 순서
    int number_data = patterns[number];

    // 자릿수 데이터와 숫자 데이터로 16비트 데이터 조합
    int output_data = (position_data << 8) | number_data;

    // MCP23017로 출력
    mcp.writeGPIOAB(output_data);
}

void loop() {
    for(int i = 1; i <= 4; i++){ // 자릿수
        for(int j = 0; j <= 9; j++){ // 숫자
            show_digit(i, j);        // i 번째 자리에 숫자 j 출력
            delay(300);
        }
    }
}
```

```

    }
  }
}

```

이제 4자리 7세그먼트 표시 장치에 분과 초가 표시되는 시계를 표시해 보자. 시작 시간은 12 분 34초로 하고 초 단위로 시간이 갱신되도록 스케치를 작성해 보자.

스케치 6. MCP23017에 연결된 4자리 7세그먼트에 시간 표시

```

#include <Wire.h>
#include <Adafruit_MCP23017.h>

// 0에서 9까지 숫자 표현을 위한
// 세그먼트 a, b, c, d, e, f, g, dp의 패턴
byte patterns[] = {
  0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE4, 0xFE, 0xE6
};

Adafruit_MCP23017 mcp;          // MCP23017 칩 객체

int SEGMENT_DELAY = 5;          // 숫자 표시 사이의 시간 간격
int minutes = 12;                // 분 초기화
int seconds = 34;                // 초 초기화

int time_previous, time_current;

void setup() {
  mcp.begin();                  // 기본 0x20 주소 사용

  for(int i = 0; i < 16; i++){
    mcp.pinMode(i, OUTPUT);      // 버튼이 연결된 핀을 입력으로 설정
  }
  time_previous = millis();
}

```

```
// 해당 자리에 숫자 하나를 표시하는 함수
void show_digit(int pos, int number){    // (위치, 출력할 숫자)
    // 상위 4비트는 사용되지 않는다.
    int position_data = 0b00001111;
    // 해당 자리의 제어 선에만 LOW 값을 가하므로 해당 비트를 0으로 만든다.
    bitClear(position_data, (pos - 1));

    // 숫자 데이터. 세그먼트 a부터 세그먼트 dp 순서
    int number_data = patterns[number];

    // 자릿수 데이터와 숫자 데이터로 16비트 데이터 조합
    int output_data = (position_data << 8) | number_data;

    // MCP23017로 출력
    mcp.writeGPIOAB(output_data);
}

void loop() {
    time_current = millis();

    if(time_current - time_previous >= 1000){
        time_previous = time_current;

        seconds++;                // 초 증가
        if(seconds == 60){        // 60초가 되면 분 증가
            seconds = 0;
            minutes++;            // 분 증가
            if(minutes == 60){    // 60분이 되면 0으로
                minutes = 0;
            }
        }
    }

    show_digit(1, seconds % 10);
    delay(SEGMENT_DELAY);
    show_digit(2, seconds / 10);
}
```

```
    delay(SEGMENT_DELAY);  
    show_digit(3, minutes % 10);  
    delay(SEGMENT_DELAY);  
    show_digit(4, minutes / 10);  
    delay(SEGMENT_DELAY);  
}
```