

Chapter 31. 디지털 입출력 확장

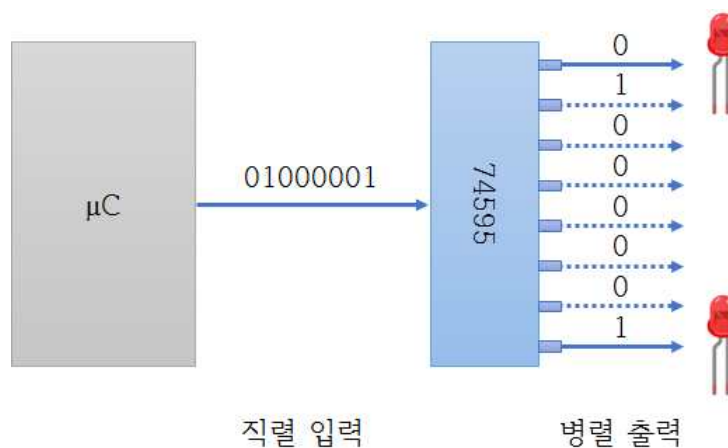
마이크로컨트롤러는 간단한 제어장치를 만들기 위해 주로 사용되므로 입출력 핀의 수가 제한적인 것이 사실이다. 하지만 가끔은 마이크로컨트롤러에 많은 수의 입력 또는 출력 장치를 연결하여 사용해야 하는 경우가 있다. 이처럼 입출력 핀의 수가 부족한 경우 사용할 수 있는 방법 중 하나가 장상 효과를 이용하는 방법이며, 다른 하나는 전용 칩을 통해 입출력을 확장하는 방법이다. 이 장에서는 입출력 핀의 수를 확장하기 위해 사용할 수 있는 다양한 전용 하드웨어의 사용 방법에 대해 알아본다.

1. 입출력 확장

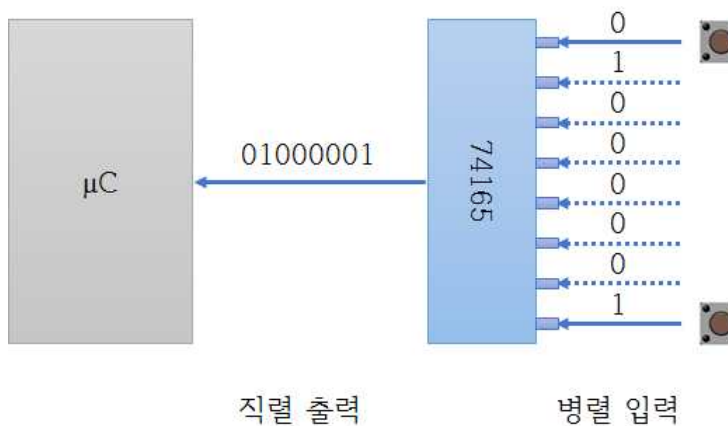
ATmega128 마이크로컨트롤러는 64개의 핀을 가지고 있으며 이 중 53개는 입출력 핀으로 사용할 수 있다. 하지만 키보드를 만든다면 100개에 달하는 푸시 버튼이 필요할 수 있고, 크리스마스 트리 장식을 만든다면 100개에 달하는 LED가 필요할 수 있으므로 53개의 입출력 핀이 그리 많지 않은 것이 사실이다. 적은 수의 핀으로 많은 수의 핀을 사용하는 것과 유사한 효과를 얻기 위한 방법으로는 4자리 7세그먼트 표시장치에서 사용한 잔상 효과가 있으며, 이는 LED를 행렬 형태로 구성한 LED 매트릭스에서도 사용된다. 또한 키 매트릭스는 4자리 7세그먼트 표시장치와 LED 매트릭스에서 사용한 잔상 효과를 입력에 적용한 경우이다. 하지만 이들 방법들은 모두 많은 수의 입출력 핀을 사용하는 것과 비슷한 효과를 내는 방법이다. 즉, 소프트웨어를 통해 비슷한 효과를 내는 방법이다. 이 장에서 살펴 볼 내용은 전용 하드웨어를 사용하여 적은 수의 핀만으로 많은 수의 핀을 사용하는 효과를 얻을 수 있는 방법들을 살펴본다.

전용 칩을 사용하여 입출력을 확장하는 방법은 UART 시리얼 통신과 기본적으로 동일하다. UART 시리얼 통신에서 데이터 전송을 위해 사용되는 핀은 RX와 TX의 2개뿐이다. UART 시리얼 통신을 사용하는 텍스트 LCD로 알파벳 대문자 'A'를 전달한다고 가정해 보자. 마이크로컨트롤러에서는 TX 핀으로 'A'에 해당하는 아스키 코드값인 65(= 0100 0001₂)를 비트 단위로 8번에 나누어 보낸다. 이 때 필요한 것이 병렬 데이터를 직렬로 변환하는 하드웨어이다. 비트 단위의 데이터를 8번에 걸쳐 수신한 텍스트 LCD는 이를 바이트 단위의 데이터인 65로 조합하여 화면에 알파벳 대문자 'A'를 표시한다. 이 과정에서 필요한 하드웨어는 직렬 데이터를 병렬로 변환하는 하드웨어이다.

직렬 데이터를 병렬로 변환하기 위해서는 74595 칩이 흔히 사용된다. 74595 칩은 직렬 입력 병렬 출력(Serial In Parallel Out, SIPO) 이동 레지스터(shift register)라고 불리며 직렬로 입력된 데이터를 병렬로 변환하여 출력하는 역할을 한다. 반면 74165 칩은 병렬 입력 직렬 출력(Parallel In Serial Out, PISO) 이동 레지스터라고 불리며 병렬로 입력된 데이터를 직렬로 변환하여 출력하는 역할을 한다. 74595 레지스터는 마이크로컨트롤러에서 직렬로 출력한 데이터를 이용하여 많은 수의 LED를 병렬로 제어하기 위해 사용할 수 있으며, 74165 레지스터는 많은 수의 버튼 입력을 직렬로 변환하여 마이크로컨트롤러로 전달하기 위해 사용할 수 있다.



(a) 74595 직렬 입력 병렬 출력 이동 레지스터 동작



(b) 74165 병렬 입력 직렬 출력 이동 레지스터 동작

그림 1. 입출력 확장을 위한 칩의 동작

74595와 74165 칩은 단방향의 데이터 전달만이 가능하므로 출력 또는 입력 확장을 위해 사

용할 수 있다. 반면 또 다른 입출력 확장 칩인 MCP23017은 양방향의 데이터 전달이 가능하므로 입력과 출력 확장을 위해 모두 사용될 수 있으며, 다른 칩과 달리 16개의 입출력 확장이 가능하다. 또한 MCP23017 칩은 I2C 통신을 사용하므로 여러 개의 MCP23107 칩을 연결하여 보다 많은 수의 입출력 확장이 가능한 장점이 있다.

2. 74595

74595는 16핀의 칩으로 하나의 데이터 입력 핀과 두 개의 제어 핀, 총 세 개의 출력 핀을 사용하여 8개의 출력을 사용할 수 있도록 해준다.

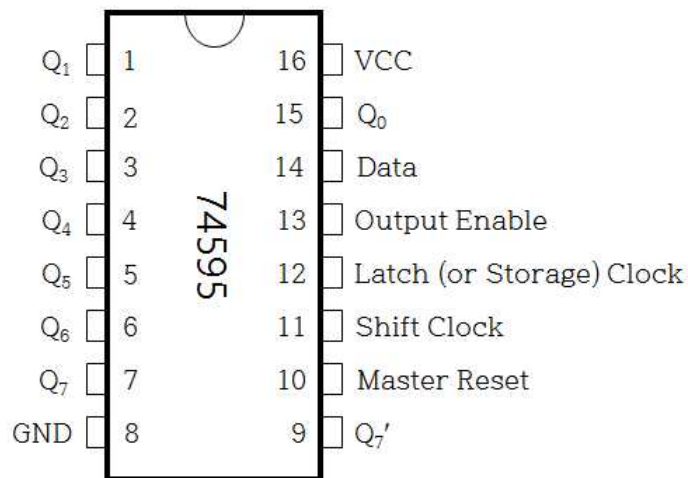


그림 2. 74595 칩 핀 배치도

74595 칩은 직렬로 1비트씩 8번에 걸쳐 8비트의 데이터를 입력 받고, 이를 병렬로 출력한다. 그림 2에서 14번 데이터 핀이 직렬 입력에 해당하며, Q₀에서 Q₇은 데이터의 병렬 출력에 해당하는 핀이다. 74595는 11번과 12번 핀에 두 개의 클록을 필요로 한다. 11번 핀의 이동 클록(shift clock)은 송신단에서 직렬로 전송하는 데이터를 수신하여 74595 내부에서 1비트씩 이동시키기 위해 사용된다. 반면 12번 래치 클록(latch clock 또는 저장장치(storage) 클록)은 74595 칩에서 병렬로 데이터가 출력되는 시점을 결정하기 위해 사용되는 클록이다. 이동 클록이 8번 발생하면 8비트의 데이터가 전송 완료되어 74595의 출력을 위한 8비트 래치에 저장되고, 이 때 래치 클록을 발생시키면 현재 래치에 저장된 데이터가 Q₀에서 Q₇ 핀을 통해 병렬로 출력된다. 출력 가능(output enable) 핀은 LOW 입력이 가해질 때 출력단의 래치를 활성화시켜 출력이 가능하도록 하는 역할을 한다. 따라서 출력 가능 핀을 GND(LOW 상태)에 연결

하고 출력 시점은 래치 클럭으로 조절함으로써 병렬 데이터 출력이 가능하다. 물론 래치 클럭을 이동 클럭과 동일하게 설정하고, 8비트 데이터가 전송된 이후 출력 가능 핀을 LOW로 설정하여 병렬 데이터 출력이 가능하도록 할 수도 있다. 이 책에서는 첫 번째 방법을 사용한다. 그림 3은 74595의 기본적인 동작 방식을 나타낸 것으로 데이터와 이동 클럭을 번갈아 8번 가한 후 마지막에 래치 클럭을 가함으로써 직렬로 전송된 8비트의 데이터가 병렬로 출력되는 과정을 보인 것이다. 이 때 출력 가능 핀은 LOW에 연결되어 있는 것으로 가정한다.

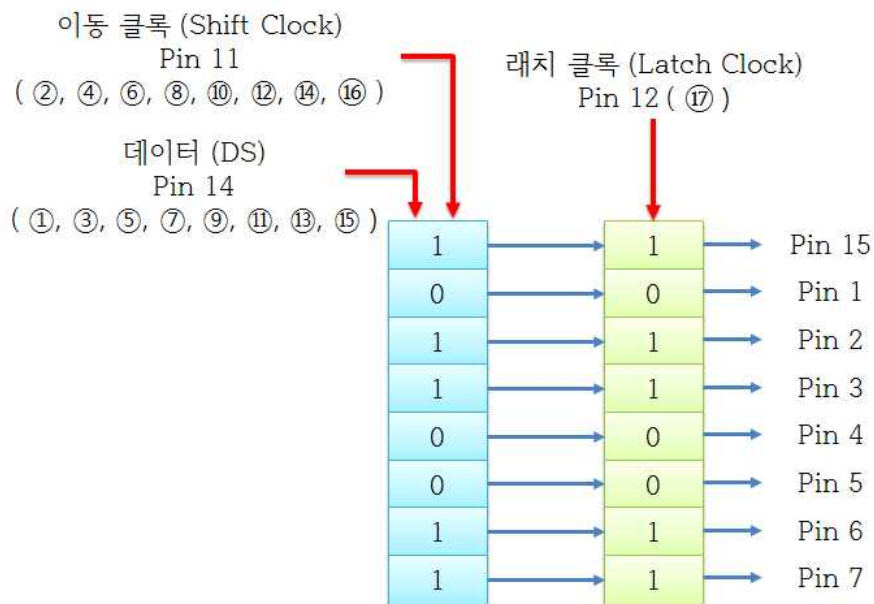


그림 3. 74595 칩의 동작 순서

10번 리셋(reset) 핀은 비동기 초기화 핀으로 VCC에 연결시키면 아무런 동작도 수행하지 않는다. 9번 Q_7' 는 쉬프트 레지스터의 값을 직렬로 출력하는 핀으로 74595를 여러 개 연결시킬 수 있도록 해준다. 9번 Q_7' 핀을 다른 74595의 DS 핀으로 연결하면 3개의 디지털 입출력 핀만으로 16개 또는 그 이상의 데이터를 출력하도록 구성할 수 있다. 표 1은 '0000 0000'으로 초기화된 쉬프트 레지스터에 8비트의 데이터(11001101, MSB 우선 전송)가 8번의 이동 클럭을 통해 이동하여 저장되는 과정을 보여준다. 이후 래치 클럭을 통해 8비트 데이터를 병렬로 출력할 수 있다.

클록	LSB Q_0	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	MSB Q_7
0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0
3	0	1	1	0	0	0	0	0
4	0	0	1	1	0	0	0	0
5	1	0	0	1	1	0	0	0
6	1	1	0	0	1	1	0	0
7	0	1	1	0	0	1	1	0
8	1	0	1	1	0	0	1	1

표 1. 74595 칩에 데이터가 저장되는 과정

74595를 ATmega128과 그림 4와 같이 포트 B의 3개 핀에 이동 클록, 래치 클록, 데이터 핀을 각각 연결한다.

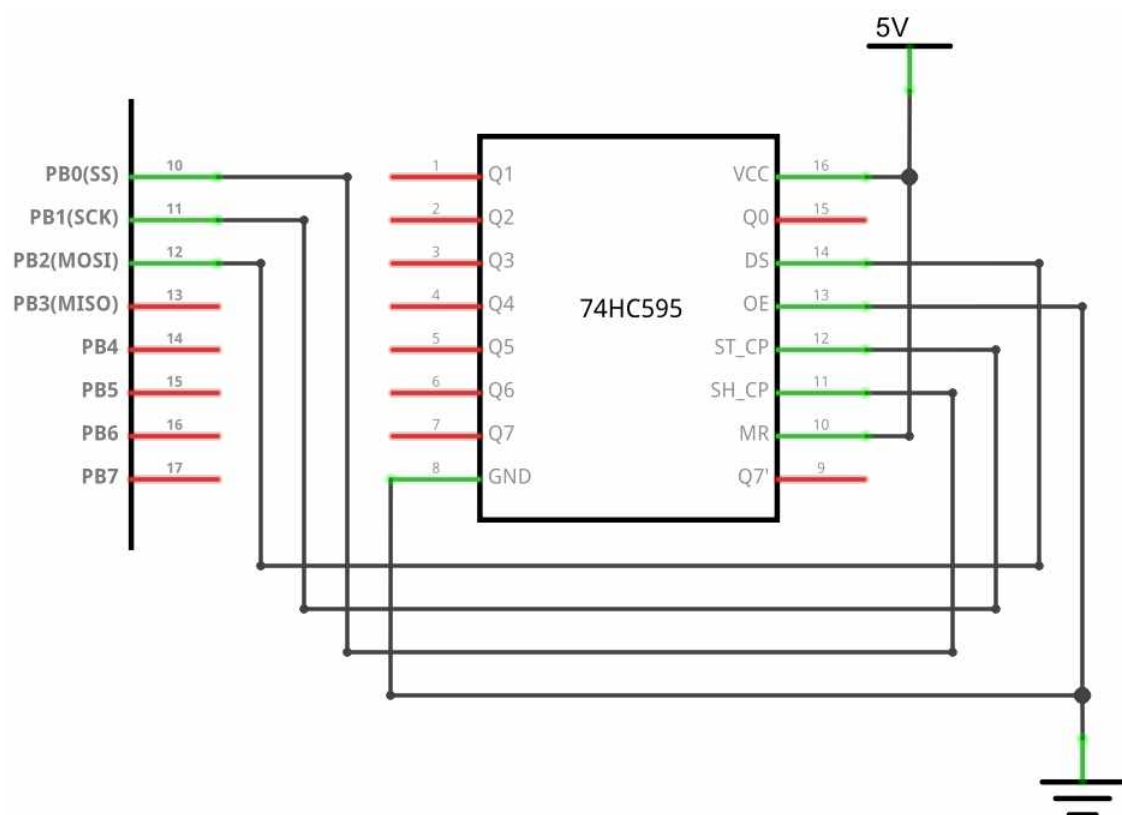


그림 4. 74595 칩 제어를 위한 연결 회로도

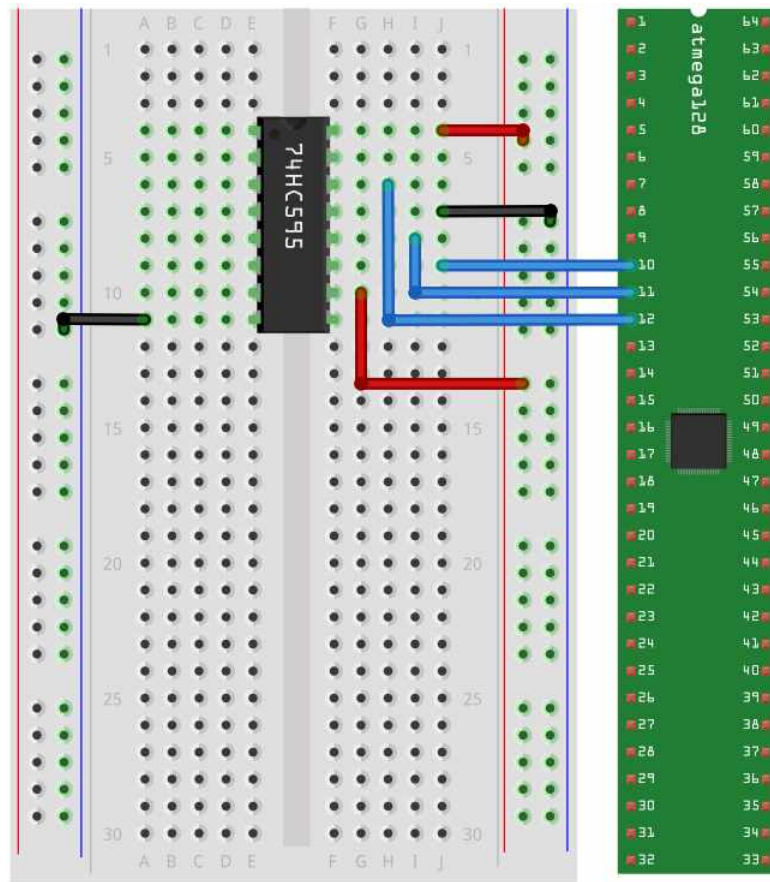


그림 5. 74595 칩 제어를 위한 연결 회로

74595 칩에는 8개의 LED를 연결한다.

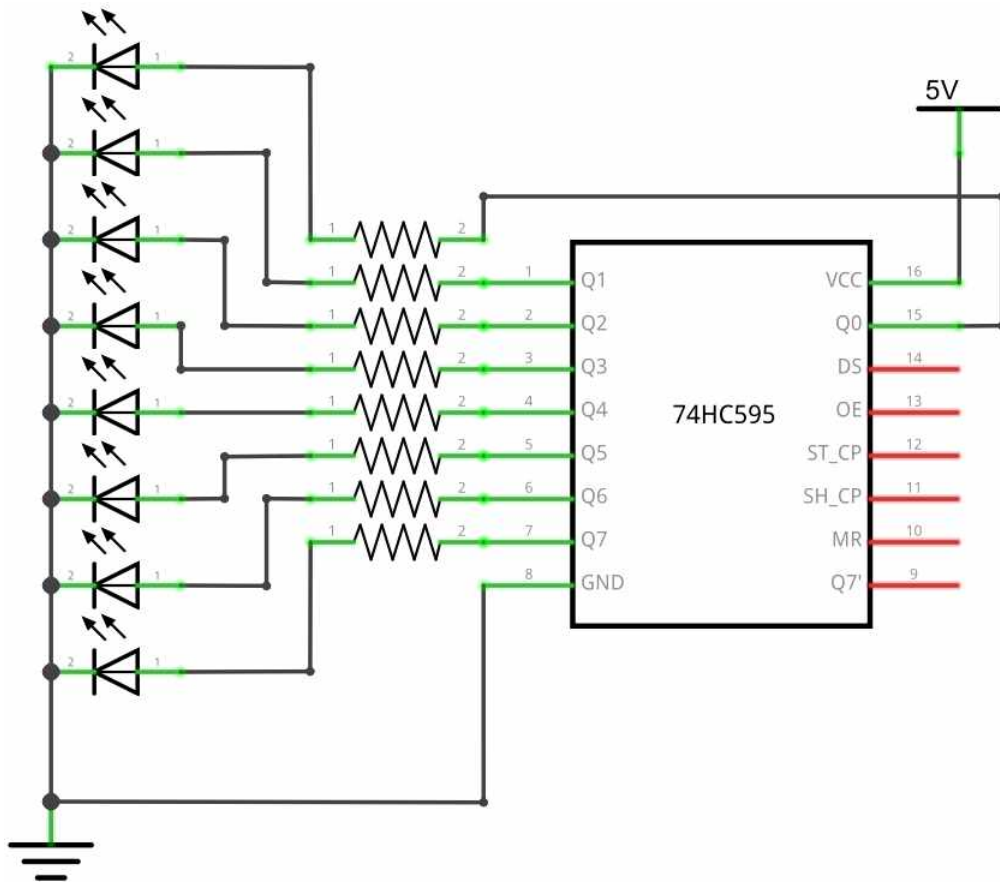


그림 6. 74595 칩과 LED 연결 회로도

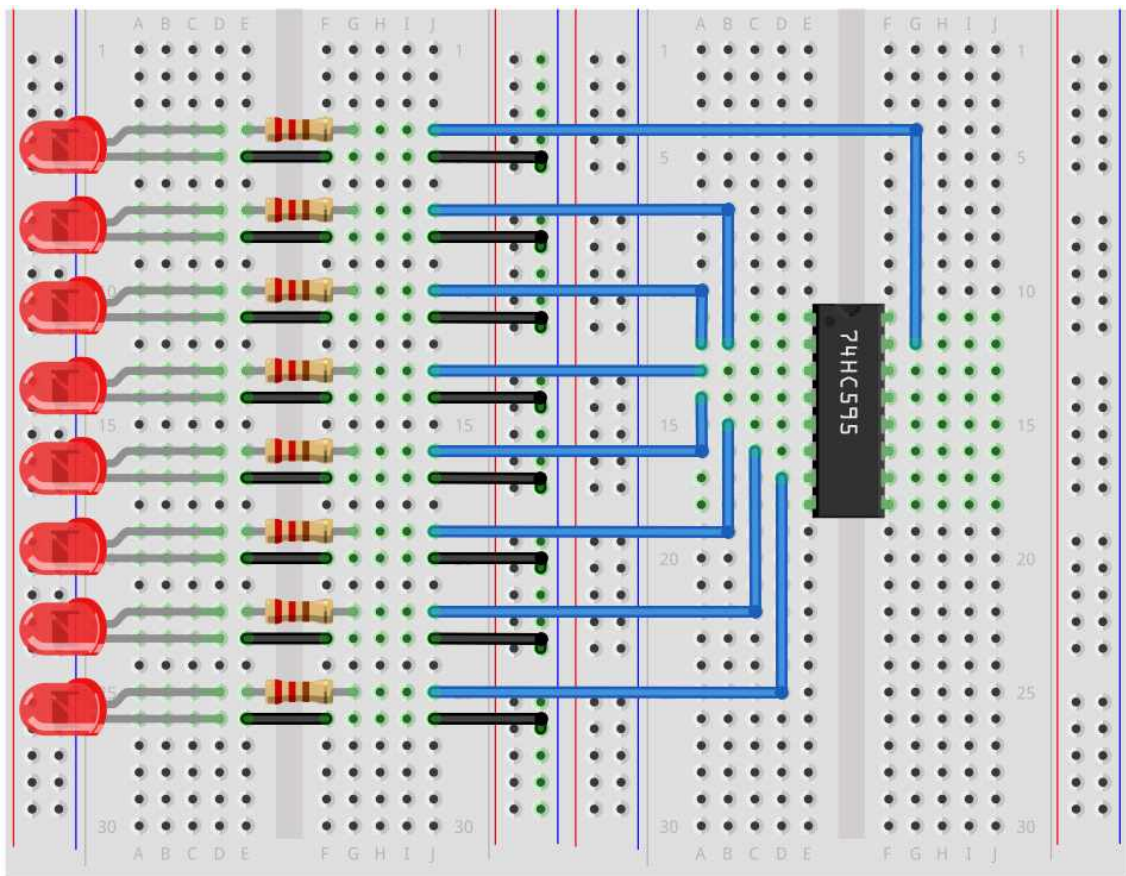


그림 7. 74595 칩과 LED 연결 회로

코드 1은 ATmega128의 3개 디지털 출력 핀을 사용하여 8개의 LED를 제어하는 예이다. LED에 표시되는 패턴은 그림 8과 같이 LED가 켜지는 위치가 1초 간격으로 변하도록 하였다.

비트 번호	7	6	5	4	3	2	1	0
패턴 1								1
패턴 2							1	
패턴 3						1		
패턴 4					1			
패턴 5				1				
패턴 6			1					
패턴 7		1						
패턴 8	1							

그림 8. LED 표시 패턴 ( : LED ON,  : LED OFF)

코드 1에서 ShiftClock 함수와 LatchClock 함수는 해당 핀의 상태를 바꾸어 클록이 발생하도록 함으로써 데이터 이동이 발생하도록 해주는 역할을 한다. 이동 클록과 래치 클록의 상승 에지(low-to-high transition)에서 데이터 이동이 발생하므로 ShiftClock 함수와 LatchClock 함수 내에서는 클록을 먼저 HIGH 상태로 설정한 후 LOW 상태로 바꾸고 있다. 실제 데이터를 전달하는 함수는 ByteDataWrite 함수로, 1 비트를 전송한 이후에는 ShiftClock 함수를 호출하여 74595 칩 내에서 데이터의 이동이 발생하도록 하고, 8 비트를 전송한 이후에는 LatchClock 함수를 호출하여 데이터가 출력되도록 하였다. ByteDataWrite 함수의 동작은 그림 2를 참고하면 된다.

코드 1. 74595 칩으로 8개 LED 제어

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>

// 'bit' 위치의 비트를 1 또는 0으로 설정하기 위한 매크로 함수
#define set_bit(bit)      ( PORTB |= _BV(bit) )
#define clear_bit(bit)    ( PORTB &= ~_BV(bit) )

// ATmega128의 포트 B에 연결된 위치
#define SHIFT_CLOCK      0
#define LATCH_CLOCK      1
#define DATA             2

void ShiftClock(void)
{
    set_bit(SHIFT_CLOCK);           // 이동 클록을 HIGH로
    clear_bit(SHIFT_CLOCK);         // 이동 클록을 LOW로
}

void LatchClock(void)
{

```

```
set_bit(LATCH_CLOCK);           // 래치 클록을 HIGH로
clear_bit(LATCH_CLOCK);         // 래치 클록을 LOW로
}

void ByteDataWrite(uint8_t data) // 1 바이트 데이터 출력
{
    for(uint8_t i = 0; i < 8; i++){
        if(data & 0b10000000) // MSB부터 1비트 출력
            set_bit(DATA);
        else
            clear_bit(DATA);

        ShiftClock();           // 1비트 출력 후 비트 이동
        data = data << 1;       // 다음 출력할 비트를 MSB로 이동
    }

    LatchClock();               // 1바이트 전달 후 실제 출력 발생
}

int main(void)
{
    // 제어 및 데이터 핀을 출력으로 설정
    DDRB |= _BV(SHIFT_CLOCK) | _BV(LATCH_CLOCK) | _BV(DATA);

    uint8_t index = 0;          // 커질 LED의 위치
    while (1)
    {
        uint8_t pattern = 1 << index; // 출력 패턴 결정
        index = (index + 1) % 8;       // 출력 패턴에서의 위치 결정

        ByteDataWrite(pattern);        // 바이트 데이터 출력

        _delay_ms(1000);
    }

    return 0;
}
```

}

3. 74165

74165 역시 16핀의 칩으로 하나의 데이터 입력 핀과 두 개의 제어 핀, 총 세 개의 핀을 사용하여 8개의 입력을 사용할 수 있도록 해준다. 74595의 경우 세 개의 핀이 모두 출력으로 설정되었지만, 74165의 경우에는 데이터 입력 용도로 사용하므로 제어 핀 2개는 출력으로 설정하지만 데이터 입력을 위한 핀은 입력으로 설정하여야 한다.

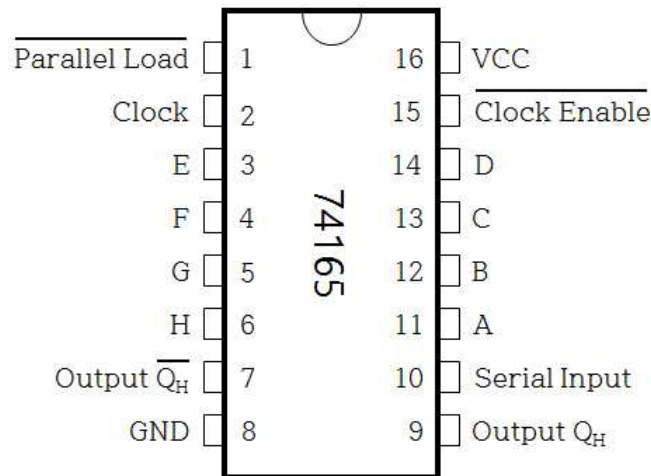


그림 9. 74165 핀 배치도

74165 이동 레지스터는 내부에 8개의 플립플롭을 가지고 있어 각 플립플롭에는 현재 입력이 저장된다. 각 입력이 가해지는 핀은 11번부터 14번까지의 핀과 3번부터 6번까지의 핀으로 이 장에서는 8개의 버튼을 연결하여 사용한다. 8개의 입력(A ~ H)이 직렬로 출력되는 핀은 7번 $\overline{Q_H}$ 와 9번 Q_H 핀으로 서로 반대되는 값이 출력된다. 직렬 입력(Serial Input) 핀은 여러 개의 74165 칩을 연결하는 경우 다른 74165 칩의 출력을 연결하는 용도로 사용할 수 있다. 하나의 74165 칩을 사용하는 경우에는 연결하지 않아도 된다.

74165 칩은 8개의 입력을 플립플롭에 저장하는 단계와 저장된 8비트의 값을 직렬로 출력하는 두 단계를 거쳐 사용한다. 저장 단계는 병렬 저장(Parallel Load) 핀에 LOW 값이 가해질 때 발생하므로 평소에는 병렬 저장 핀에 HIGH 값을 가하고, 데이터를 입력하는 시점에서 LOW 값으로 변경하면 된다.

직렬 출력 단계는 병렬 저장 핀에 HIGH 값이 가해지고, 클록 활성화 (Clock Enable) 핀에 LOW 값이 가해진 상태어야 한다. 클록 활성화 핀에 HIGH 값이 가해지면 74165 칩의 직렬 출력을 사용할 수 없으며, 저장 단계는 클록 활성화 핀의 값과 무관하게 동작하므로 클록 활성화 핀은 GND에 연결하여 사용할 수 있다. 비트 단위의 데이터 전송은 클록 핀의 값이 LOW에서 HIGH로 변하는 상향 에지에서 발생한다.

먼저 74165 칩을 ATmega128과 연결하자. 포트 B의 3개 핀에 병렬 저장, 클록, 직렬 출력 핀을 각각 연결한다.

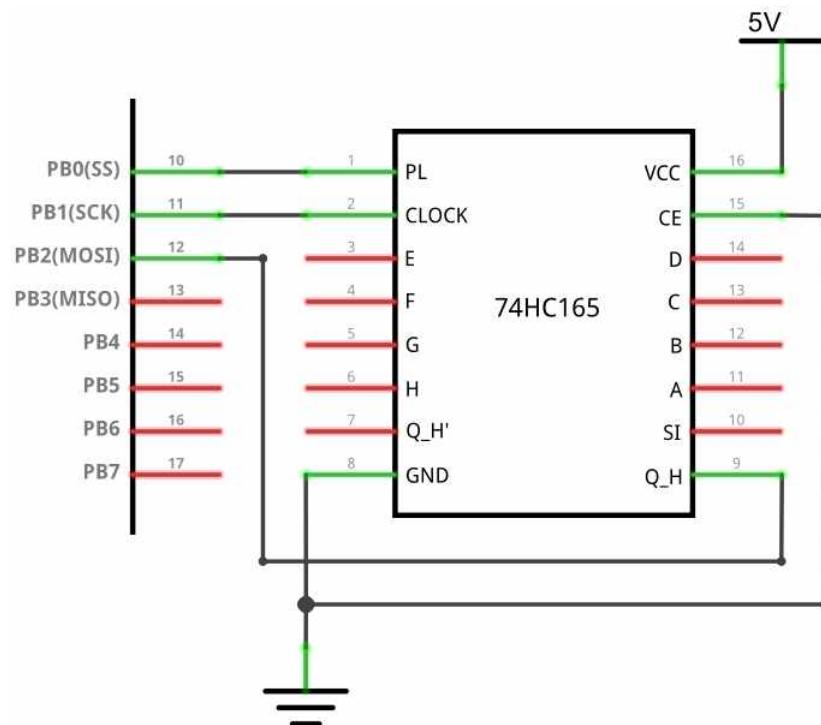


그림 10. 74165 칩 제어를 위한 연결 회로도

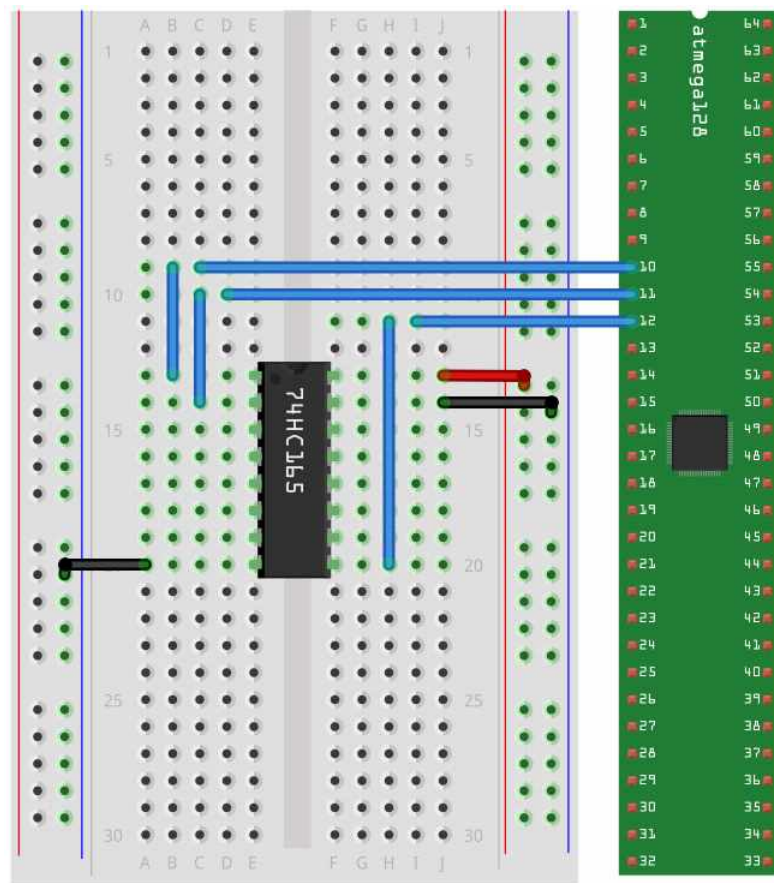


그림 11. 74165 칩 제어를 위한 연결 회로

74165 칩에는 8개의 버튼을 풀다운 저항을 사용하여 연결한다.

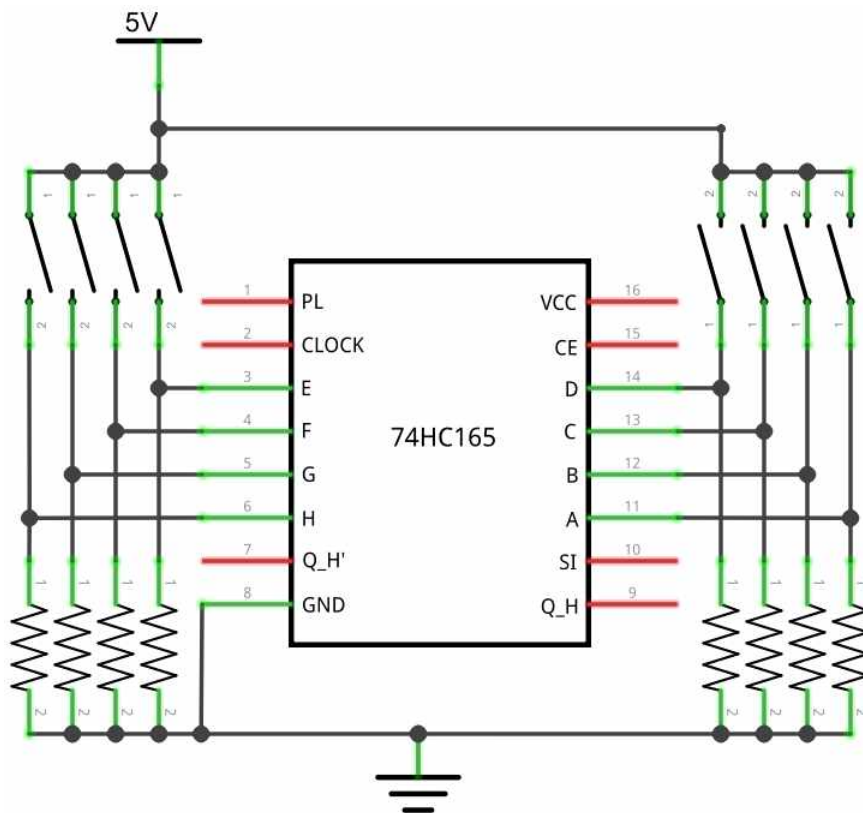


그림 12. 74165 칩과 버튼 연결 회로도

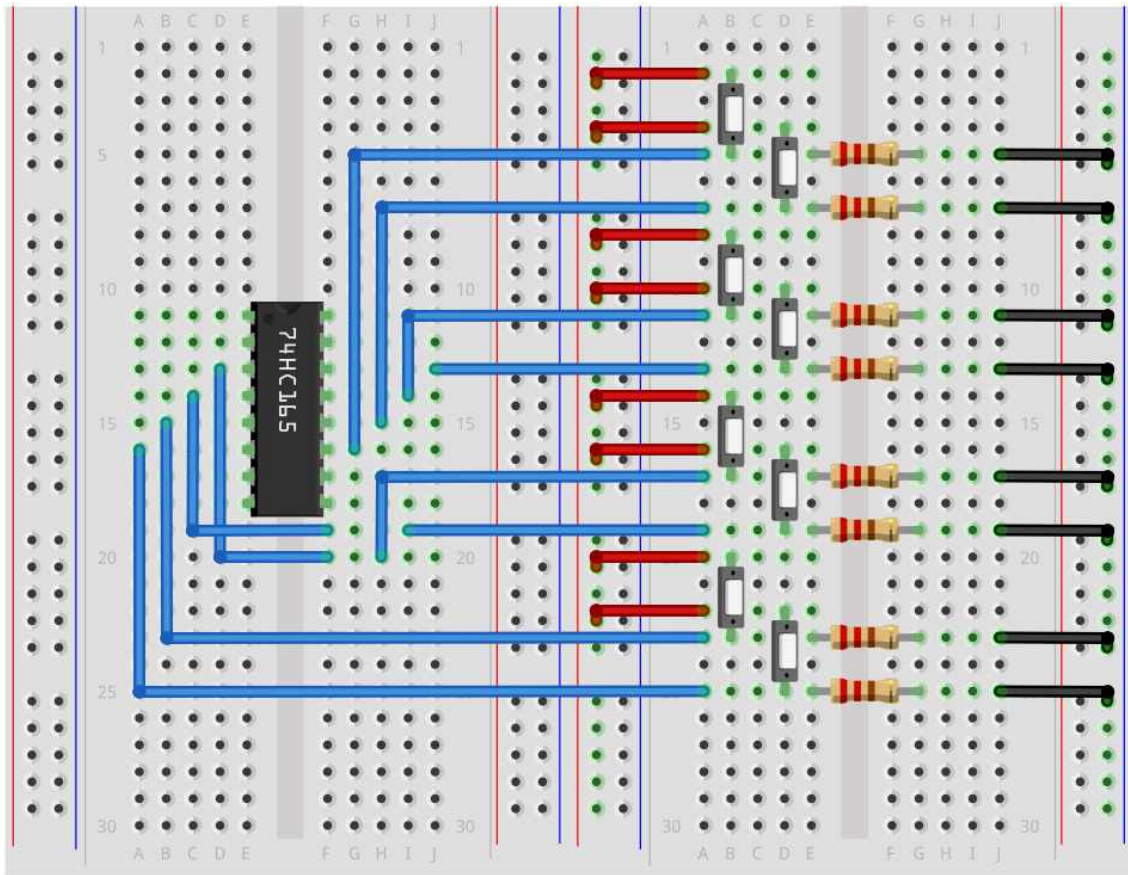


그림 13. 74165 칩과 버튼 연결 회로

코드 2는 ATmega128의 3개 디지털 입출력 핀을 사용하여 8개의 버튼 입력을 받는 코드의 예이다. 코드 2에서는 8번에 걸쳐 74165 칩 내부의 플립플롭에 저장된 각 버튼의 상태를 읽어 터미널 프로그램으로 출력하고 있다.

코드 2. 74165 칩으로 8개 버튼 제어

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
#include "UART1.h"

// 'bit' 위치의 비트를 1 또는 0으로 설정하기 위한 매크로 함수
#define set_bit(bit)      ( PORTB |= _BV(bit) )
```

```
#define clear_bit(bit)    ( PORTB &= ~_BV(bit) )

// ATmega128의 포트 B에 연결된 위치
#define PARALLEL_LOAD    0
#define CLK              1
#define SERIAL_OUTPUT    2

FILE OUTPUT \
    = FDEV_SETUP_STREAM(UART1_transmit, NULL, _FDEV_SETUP_WRITE);
FILE INPUT \
    = FDEV_SETUP_STREAM(NULL, UART1_receive, _FDEV_SETUP_READ);

int main(void)
{
    stdout = &OUTPUT;
    stdin = &INPUT;

    UART1_init();                // UART 통신 초기화

    // 병렬 입력과 클록 핀을 출력으로 설정
    DDRB |= _BV(PARALLEL_LOAD) | _BV(CLK);

    while (1)
    {
        // 저장 단계
        clear_bit(PARALLEL_LOAD);
        _delay_ms(5);

        set_bit(PARALLEL_LOAD);
        _delay_ms(5);

        // 직렬 출력 단계
        uint8_t button;
        for(int i = 0; i < 8; i++){
            button = (PINB & _BV(SERIAL_OUTPUT)) >> SERIAL_OUTPUT;
            printf("%c ", (button == 1) ? 'O' : 'X');
```

```
// 상승 에지에서 직렬로 1비트씩 입력
clear_bit(CLK);
_delay_ms(1);
set_bit(CLK);
_delay_ms(1);
}
printf("\r\n");

_delay_ms(1000);          // 1초에 한 번 버튼 검사
}

return 0;
}
```

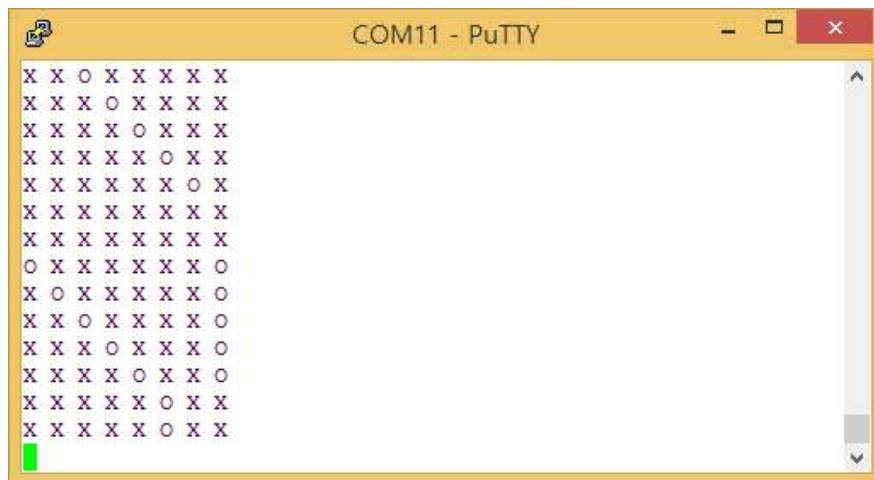


그림 14. 코드 2 실행 결과

4. MCP23017

74595와 74165 칩이 3개의 입출력 핀으로 8개의 출력과 입력을 사용할 수 있도록 해주는 칩이라면, MCP23017 칩은 2개의 입출력 핀만을 사용하여 16개의 입출력을 사용할 수 있도록 해주는 칩에 해당한다. MCP23017 칩이 2개의 입출력 핀만을 사용하는 것은 I2C 통신을 사용하기 때문이다. I2C 통신을 사용하는 장치는 주소가 다르다면 여러 개의 장치를 2개의 연결선

만을 사용하여 연결할 수 있다. MCP23017의 경우 최대 8개까지 동시에 연결하여 사용할 수 있으므로 최대 128개($= 16 \times 8$)의 추가적인 입출력 핀을 사용할 수 있도록 해준다. MCP23017은 28개의 핀을 가지며 16비트의 입출력을 뱅크(bank)라 불리는 8비트 단위의 두 그룹으로 나누어 관리한다.

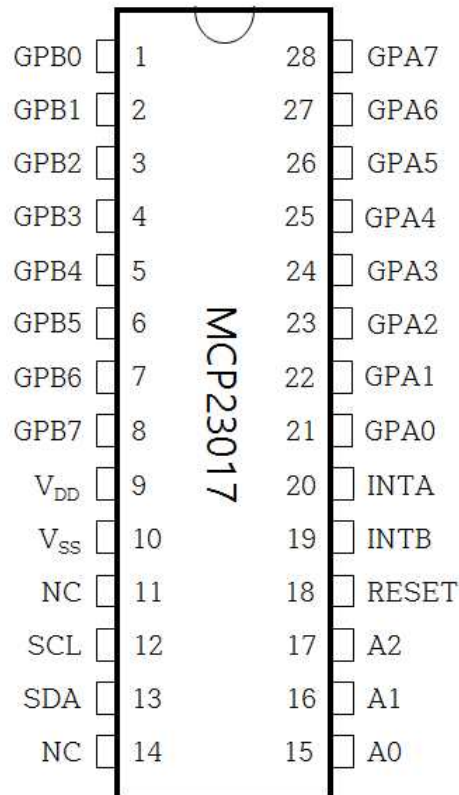


그림 15. MCP23017 칩 핀 배치도

9번 핀 V_{DD}는 5V에, 10번 핀 V_{SS}는 GND에 연결하는 전원 핀이다. 11번과 14번 핀은 사용하지 않으며 (NC, Not Connected), 12번과 13번 각각 I2C 연결을 위한 클록(SCL)과 데이터(SDA) 핀으로 ATmega128의 PD0(SCL)와 PD1(SDA) 핀에 각각 연결한다. I2C 연결에서는 안정적인 데이터 전송을 위해 외부 풀업저항 연결이 필요하며 일반적으로 10kΩ 저항을 풀업 저항으로 사용한다. 18번 RESET 핀은 GND에 연결하여 칩을 초기화하기 위해 사용하는 핀으로 5V에 연결하여 아무런 동작을 수행하지 않는다. 19번과 20번 핀은 인터럽트 관련 핀으로 여기서는 사용하지 않는다. 마지막 15번에서 17번까지의 핀은 I2C 통신을 위한 주소 지정에 사용된다. 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 $0010\ 0000_2 + 0000\ 0100_2 = 0010\ 0100_2 = 0x24$ 가 된다. 이처럼 MCP23017 칩에는 $0x20$ 에서 $0x27$ 까지 8개의 서로 다른 주소를 지정할 수 있으므로 최대 8개까지 연결하여 사용할 수 있다. 여기서는 기본 주소인 $0x20$ 을 사용한다.

MCP23017을 ATmega128에 그림 16과 같이 연결하자. MCP23017은 I2C 통신을 사용하므로 반드시 I2C 통신을 위해 지정된 핀에 연결하여야 한다.

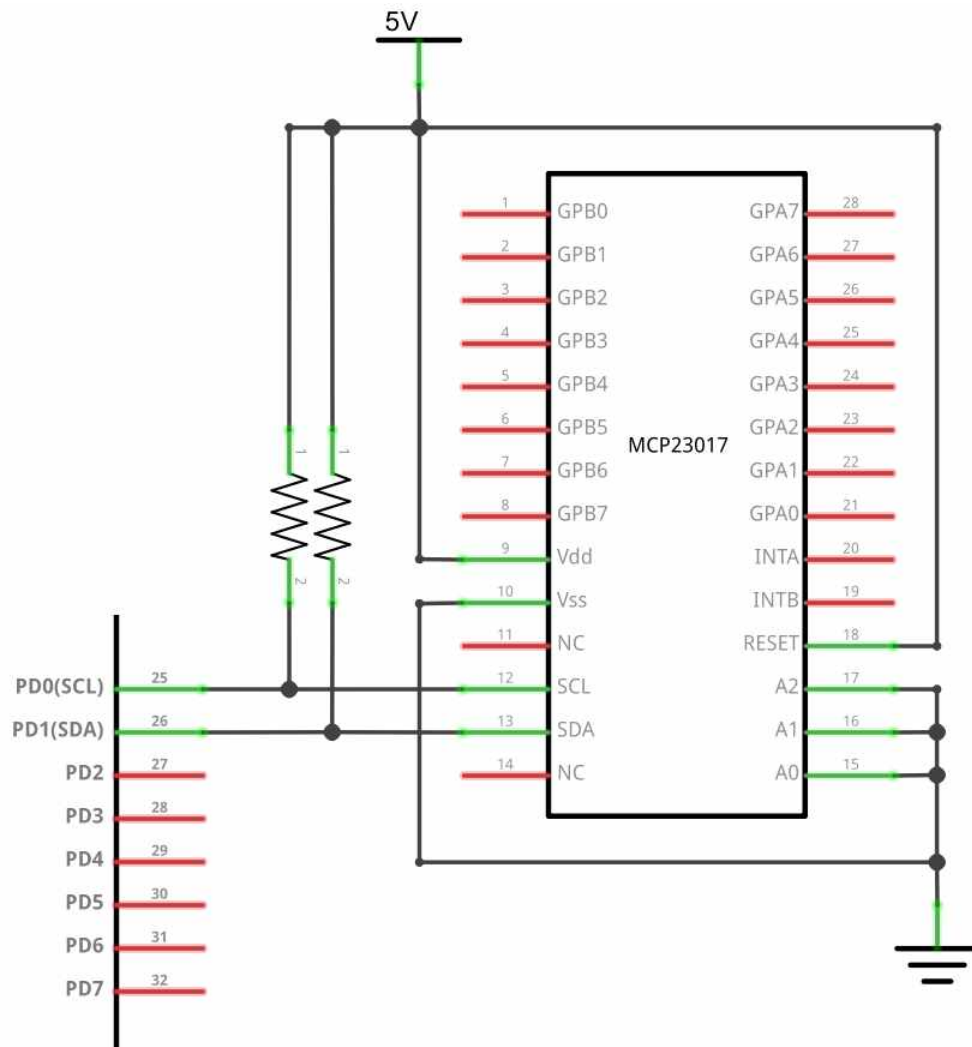


그림 16. MCP23017 칩 제어를 위한 연결 회로도

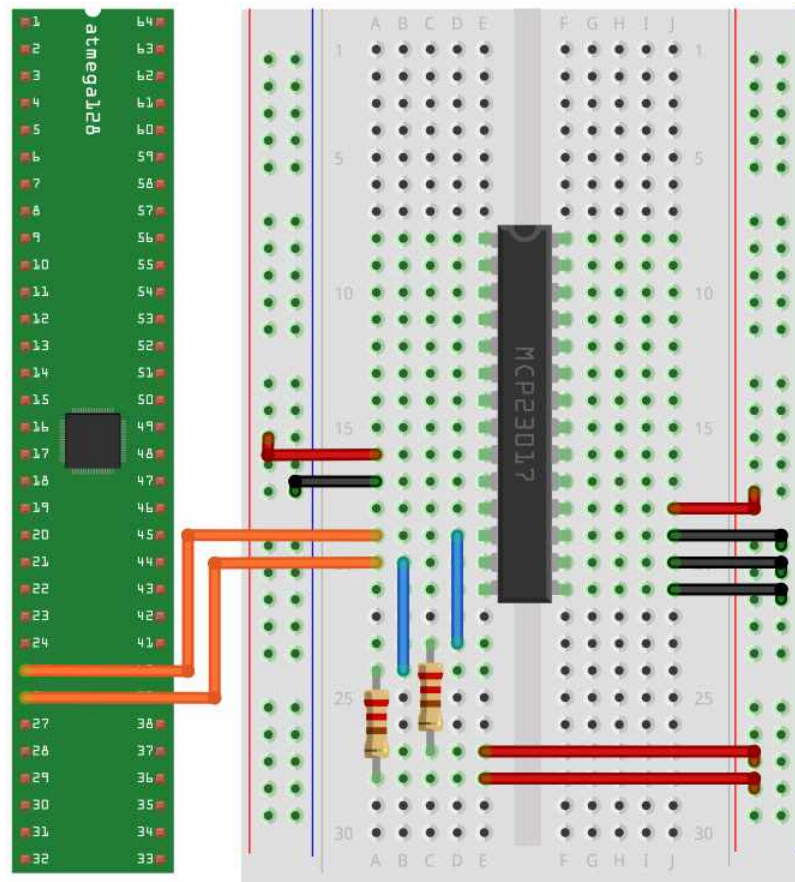


그림 17. MCP23017 칩 제어를 위한 연결 회로

MCP23017 칩의 뱅크 A에는 8개의 LED를 연결하고, 뱅크 B에는 8개의 버튼을 풀다운 저항을 사용하여 연결한다. LED와 버튼 연결은 그림 6과 그림 12를 참고하여 연결하면 된다.

MCP23017 칩으로 입출력을 확장하기 위해서는 입출력 설정을 위한 IODIR 레지스터, 입력으로 사용되는 경우 풀업 저항 사용을 제어하기 위한 GPPU 레지스터, 입출력 데이터가 저장되는 GPIO 레지스터 등을 사용하여야 한다. 표 2는 기본적인 디지털 데이터 입출력을 위해 필요한 레지스터 이름과 레지스터 주소를 요약한 것이다.

레지스터	주소	설명
IODIRA	0x00	뱅크 A 입출력 설정 레지스터
IODIRB	0x01	뱅크 B 입출력 설정 레지스터
GPPUA	0x0C	뱅크 A 풀업저항 설정 레지스터
GPPUB	0x0D	뱅크 B 풀업저항 설정 레지스터
GPIOA	0x12	뱅크 A 데이터 레지스터
GPIOB	0x13	뱅크 B 데이터 레지스터

표 2. MCP23017 레지스터

코드 3은 뱅크 A에 연결된 LED를 뱅크 B에 연결된 버튼으로 제어하는 코드의 예로, 8개 중 눌린 버튼의 위치에 대응하는 LED가 켜진다. I2C 통신을 위한 기본적인 함수들은 I2C 장에서 RTC 제어를 위해 만들었던 I2C 라이브러리를 사용하였으며, 출력 결과를 터미널에 표시하기 위해서 UART1 라이브러리를 사용하였다. I2C 통신에 관한 자세한 내용은 **I2C 장**을 참고하면 된다.

코드 3. MCP23017 칩으로 8개 버튼과 8개 LED 제어

```
#define F_CPU 16000000L
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include "I2C_RTC.h"
#include "UART1.h"

FILE OUTPUT \
= FDEV_SETUP_STREAM(UART1_transmit, NULL, _FDEV_SETUP_WRITE);
FILE INPUT \
= FDEV_SETUP_STREAM(NULL, UART1_receive, _FDEV_SETUP_READ);

#define MCP23017_ADDRESS      0x20
#define MCP23017_IODIRA      0x00
#define MCP23017_IODIRB      0x01
#define MCP23017_GPPUA       0x0C
#define MCP23017_GPPUB       0x0D
```

```
#define MCP23017_GPIOA          0x12
#define MCP23017_GPIOB          0x13

int main(void)
{
    I2C_init();                // I2C 통신
    UART1_init();              // UART 통신 초기화

    stdout = &OUTPUT;
    stdin = &INPUT;

    I2C_start();                // I2C 시작
    I2C_transmit(MCP23017_ADDRESS << 1); // I2C 주소 전송. 쓰기 모드
    I2C_transmit(MCP23017_IODIRA);      // IODIRA 레지스터 주소
    I2C_transmit(0x00);                // बैंक A 핀을 출력으로 설정
    I2C_transmit(0xFF);                // बैंक B 핀을 입력으로 설정
    I2C_stop();                        // I2C 정지

    I2C_start();
    I2C_transmit(MCP23017_ADDRESS << 1); // I2C 주소 전송. 쓰기 모드
    I2C_transmit(MCP23017_GPPUB);        // GPPUB 레지스터 주소
    I2C_transmit(0);                     // बैंक B 핀의 풀업 저항 해제
    I2C_stop();

    while(1)
    {
        // बैंक B의 버튼 상태 읽기
        I2C_start();
        I2C_transmit(MCP23017_ADDRESS << 1); // I2C 주소 전송. 쓰기 모드
        I2C_transmit(MCP23017_GPIOB);        // GPIOB 레지스터 주소
        I2C_stop();

        I2C_start();
        I2C_transmit((MCP23017_ADDRESS << 1) + 1); // I2C 주소 전송. 읽기 모드
        uint8_t button_state = I2C_receive_NACK(); // 버튼 상태 읽기
        I2C_stop();
    }
}
```

```

printf("%d\r\n", button_state);           // 버튼 상태 출력

// बैं크 A의 LED에 बैं크 B의 버튼 상태 출력
I2C_start();
I2C_transmit(MCP23017_ADDRESS << 1);      // I2C 주소 전송. 쓰기 모드
I2C_transmit(MCP23017_GPIOA);             // GPIOA 레지스터 주소
I2C_transmit(button_state);               // 버튼 상태를 LED로 출력
I2C_stop();

_delay_ms(500);
}

return 0;
}

```

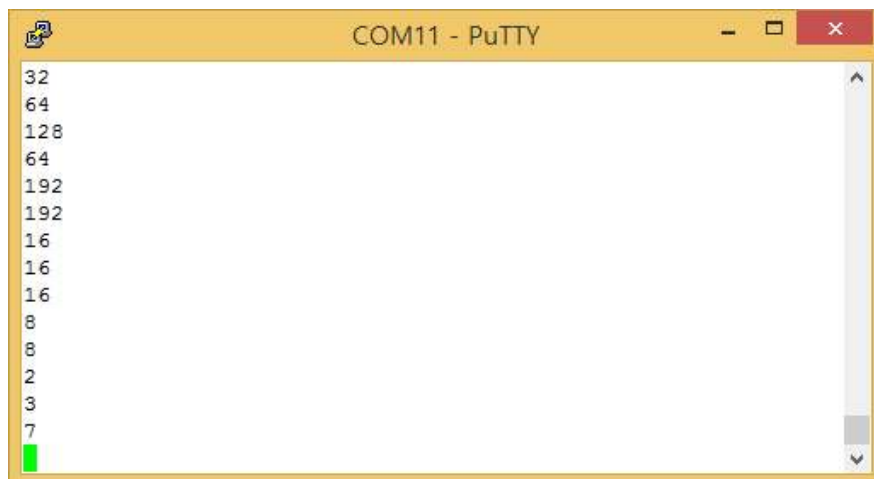


그림 18. 코드 3 실행 결과

5. 맺는말

마이크로컨트롤러는 간단한 제어장치를 만들기 위해 주로 사용되므로 많은 수의 입출력 장치를 연결하는 경우를 흔히 볼 수 있지는 않다. 하지만 많은 수의 LED나 버튼을 사용하고자 하는 경우에는 마이크로컨트롤러의 입출력 핀이 많지 않다는 것을 쉽게 알아차릴 수 있다. 입출력 핀이 부족한 경우 확장할 수 있는 방법은 소프트웨어적인 방법과 하드웨어적인 방법이 있

을 수 있다. 소프트웨어적인 방법은 4자리 7세그먼트 표시장치를 위해 사용하였던 잔상효과가 대표적이다. 또한 이와 유사하게 많은 수의 버튼을 제어하는 것도 가능하다. 전용의 하드웨어를 사용한다면 보다 안정적으로 입출력을 확장할 수 있으며 이 장에서 살펴본 입출력 확장을 위한 전용 칩들이 이에 해당한다. 디지털 출력을 확장하기 위한 74595 칩, 디지털 입력을 확장하기 위한 74165 칩은 단방향의 데이터 흐름만을 지원하므로 입력이나 출력 전용으로만 사용할 수 있다. 반면 MCP23017 칩은 양방향의 데이터 흐름을 지원하므로 입출력 확장에 모두 사용할 수 있으며, I2C 통신을 사용하므로 확장성이 뛰어나다는 점 등은 장점이라 할 수 있다. 이 장에서는 마이크로컨트롤러에서 기본이 되는 디지털 입출력의 확장만을 다루었지만, 아날로그 입력 핀 역시 744051 칩 등을 통해 확장할 수 있으므로 필요한 경우 고려해볼 수 있다.

연습 문제

1. 코드 1은 74595 칩을 사용하여 8개의 LED를 제어하는 예이다. 8개의 LED 대신 1자리 7세그먼트 표시장치를 74595 칩의 병렬 출력에 연결하고 0.5초 간격에서 0에서 9까지의 숫자가 반복하여 출력되도록 코드를 수정해 보자.
2. 코드 3은 버튼이 눌려진 경우 눌려진 버튼에 해당하는 LED만 켜지고 버튼을 떼면 LED는 꺼진다. 이를 다음번 버튼이 눌려질 때까지 LED가 켜져 있도록 수정해 보자.
3. 코드 3과 문제 2번에서는 버튼이 눌려지면 그 위치의 LED만 켜진다. 이를 버튼이 눌려지면 눌려진 버튼의 위치까지 LED가 켜지고, 2번 문제에서와 같이 다음 버튼이 눌려질 때까지 이를 유지하도록 수정해 보자. 예를 들어 뱅크 B의 n ($n = 0, \dots, 7$)번 버튼이 눌려지면 뱅크 A의 $n + 1$ 개 LED가 켜진다. 버튼 여러 개가 한꺼번에 눌려지는 경우에는 가장 큰 n 값에 의해 켜지는 LED의 수를 결정하도록 한다.