

Chapter 221. 아두이노-아두이노 SPI 연결

SPI(Serial Peripheral Interface)는 고속의 주변장치 연결을 위한 동기식, 전이중 방식의 시리얼 통신 방식으로 1:N의 연결을 지원하고, SS(Slave Select) 연결선을 통해 여러 개의 주변장치 중 하나를 선택하는 하드웨어적인 방식을 사용하고 있다. 흔히 비교되는 I2C(Inter-Integrated Circuit)는 반이중 방식을 사용하는 저속의 시리얼 통신 방식으로 주변장치의 주소를 사용한 소프트웨어적인 방식으로 여러 개의 주변장치 중 하나를 선택하는 차이가 있다.

SPI 통신을 위해서는 데이터 송수신을 위해 MISO(Master In Slave Out)와 MOSI(Master Out Slave In)를, 동기화 클럭을 위해 SCK(Serial Clock), 그리고 주변장치 선택을 위해 SS를 사용하여 4-Wire 라고도 불린다. SPI는 장치 선택을 위해 SS 연결선을 사용하기 때문에 I2C에서와 같은 주소 충돌이 발생하지 않는 장점이 있는 반면, 연결하는 주변장치의 수가 증가하면 필요한 연결선의 수가 그에 비례하여 증가하는 점은 단점이 될 수도 있다.

아두이노에서는 SPI 통신을 위해 SPI 라이브러리를 제공하고 있으므로 SPI 통신을 통해 주변장치와 손쉽게 통신이 가능하다.

1. 슬레이브로 데이터 전송

두 개의 아두이노 우노를 SPI 통신으로 연결해 보자. 두 개의 아두이노는 UART 연결에서와 마찬가지로 COM35와 COM36에 연결된 것으로 가정하며 COM35에 연결된 아두이노를 마스터, COM36에 연결된 아두이노를 슬레이브로 한다.

아두이노 우노에 사용된 ATmega328은 SPI 연결을 위해 전용 핀을 할당하고 있다. 여러 개의 주변장치를 연결하는 경우 MOSI, MISO, SCK는 공통으로 사용되며, SS는 주변장치 별로 하나씩 필요하다. 표 1은 아두이노 우노에서 SPI 연결을 위해 사용되는 핀을 나타낸 것으로, SS 핀은 기본적으로 디지털 10번 핀을 사용하도록 지정되어 있지만 필요에 따라 변경할 수 있다.

SPI 신호	Arduino UNO	비고
MOSI	디지털 11	데이터
MISO	디지털 12	데이터
SCK	디지털 13	클록
SS	디지털 10	변경 가능

표 1. SPI 연결 핀

두 개의 아두이노 우노를 그림 1과 같이 SPI를 통해 연결해 보자.

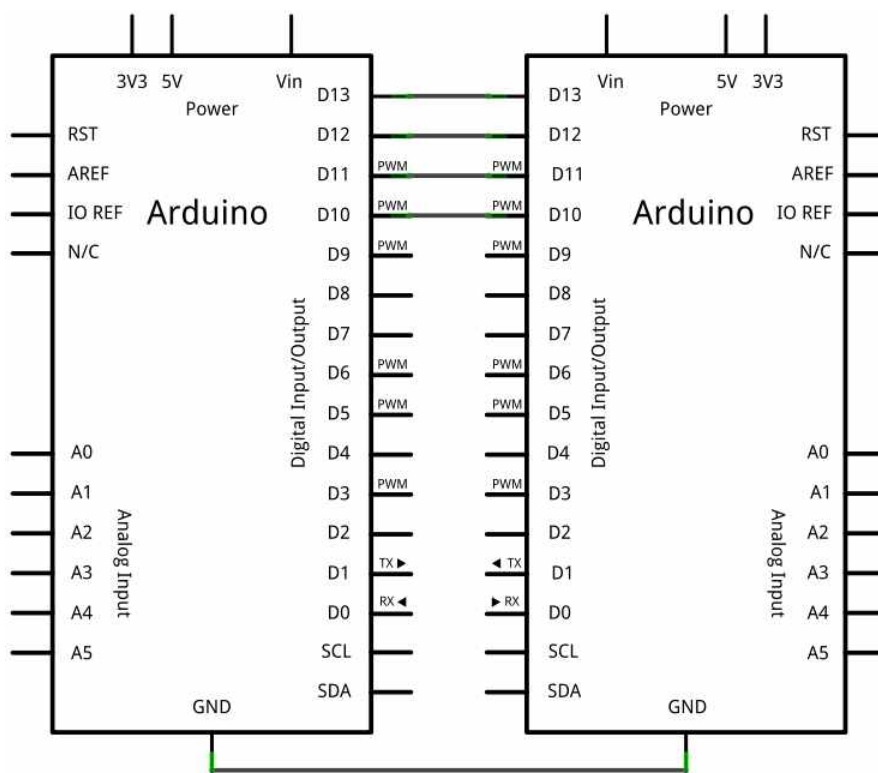


그림 1. 두 개의 아두이노 우노 사이의 SPI 연결 회로도

스케치 1은 마스터가 1초 간격으로 'Hello World!'라는 문자열을 전송하는 스케치의 예이다. 문자열을 수신하는 슬레이브는 전송된 문자열의 끝을 개행문자 '\n'으로 판별하므로 전송하는 문자열의 끝에 개행문자를 추가로 전송한다.

스케치 1. SPI 통신 테스트 - 마스터

```
#include <SPI.h>

void setup (void)
{
    SPI.begin ();                // SPI 통신 초기화
    digitalWrite(SS, HIGH);      // 슬레이브가 선택되지 않은 상태로 유지

    // 안정적인 전송을 위해 분주비를 높여 전송 속도를 낮춤
    SPI.setClockDivider(SPI_CLOCK_DIV16);
}

void loop (void)
{
    const char *p = "Hello, World\n";

    digitalWrite(SS, LOW);       // 슬레이브를 선택한다.
    for (int i = 0; i < strlen(p); i++){ // 문자열을 전송한다.
        SPI.transfer(p[i]);
    }

    digitalWrite(SS, HIGH); // 슬레이브 선택을 해제한다.

    delay(1000);
}
```

마스터의 스케치는 스케치 1과 같이 간단하게 작성할 수 있지만, 아두이노의 SPI 라이브러리는 마스터 모드만을 지원하므로 슬레이브의 경우에는 ATmega328의 SPI 통신 기능을 레지스터를 통해 직접 제어하여야 한다.

ATmega328에서 SPI는 인터럽트 방식으로 동작한다. SPI 통신을 통해 데이터가 수신되면 인터럽트 서비스 루틴(ISR, Interrupt Service Routine)이 자동으로 호출된다. 이 때 ISR의 매개변수로는 SPI 통신을 통한 데이터 수신 인터럽트에 해당하는 SPI_STC_vect가 전달된다. ATmega328이 SPI에 의한 데이터 수신 인터럽트를 처리할 수 있도록 하기 위해서는 세 가지 설정이 필요하다.

- SPI 통신을 사용 가능하도록 설정하여야 한다.
- SPI 통신에서 슬레이브로 동작하도록 설정하여야 한다.
- SPI 통신을 통해 데이터가 수신된 경우 인터럽트가 발생하도록 설정하여야 한다.

위의 세 가지 설정은 모두 SPI Control Register인 SPCR의 해당 비트를 설정해 줌으로써 가능하다. SPCR 레지스터 구조는 그림 2와 같다.

비트	7	6	5	4	3	2	1	0
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
읽기/쓰기	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
초깃값	0	0	0	0	0	0	0	0

그림 2. SPCR 레지스터 구조

SPCR 레지스터 중 6번 비트인 SPE(SPI Enable) 비트를 1로 설정함으로써 SPI 통신이 가능하다. 4번 비트인 MSTR(Master/Slave Select) 비트가 1인 경우 마스터 모드로 동작하며, 0인 경우 슬레이브 모드로 동작한다. 디폴트값은 0이다. 마지막으로 7번 비트인 SPIE(SPI Interrupt Enable) 비트를 1로 설정하면 SPI 통신에 의한 인터럽트 발생이 허용된다. MSTR, SPE, SPIE는 각각 4, 6, 7의 상수로 비트 번호를 나타내도록 정의되어 있다.

```
#define MSTR      4
#define SPE       6
#define SPIE      7
```

각 비트는 _BV 매크로를 이용한 비트 연산을 통해 설정할 수 있다. _BV 매크로는 해당 위치의 비트만을 1로 하는 마스크를 생성하는 매크로로 다음과 같이 정의되어 있다.

```
#define _BV(bit) (1 << (bit))
```

따라서 _BV 매크로를 통해 생성된 마스크와 비트 OR 연산을 수행함으로써 해당 비트만을 1

로 설정할 수 있으며, 마스크를 반전시켜 비트 AND 연산을 수행함으로써 해당 비트만을 0으로 설정할 수 있다. 스케치 2는 슬레이브가 SPI 통신을 통해 문자열을 수신하고 개행문자를 만나는 경우 시리얼 모니터로 수신된 문자열을 출력하는 함수이다.

스케치 2. SPI 통신 테스트 - 슬레이브

```
#include <SPI.h>

char buf[100];           // 수신된 문자 저장을 위한 버퍼
volatile byte pos;       // 수신 버퍼에 문자를 기록할 위치
volatile boolean process_it; // 개행 문자를 만난 경우 시리얼 모니터로 출력

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

    // SPI 통신을 위한 핀들의 입출력 설정
    pinMode(MISO, OUTPUT);
    pinMode(MOSI, INPUT);
    pinMode(SCK, INPUT);
    pinMode(SS, INPUT);

    // 마스터의 전송 속도에 맞추어 통신 속도를 설정한다.
    SPI.setClockDivider(SPI_CLOCK_DIV16);

    // SPI 통신을 위한 레지스터를 설정
    SPCR |= _BV(SPE);       // SPI 활성화
    SPCR &= ~_BV(MSTR);     // Slave 모드 선택
    SPCR |= _BV(SPIE);      // 인터럽트 허용

    pos = 0;                // 수신 문자 기록 위치 초기화
    process_it = false;

}

// SPI 통신으로 데이터가 수신될 때 발생하는 인터럽트 처리 루틴
ISR (SPI_STC_vect)
```

```
{
    byte c = SPDR;                // 수신된 문자를 얻어온다.

    if (pos < sizeof(buf)){        // 현재 버퍼에 저장할 공간이 있는 경우
        buf[pos++] = c;           // 버퍼에 수신된 문자 기록

        if (c == '\n'){           // 개행 문자를 만나면 출력
            process_it = true;
        }
    }
}

void loop (void)
{
    if (process_it){              // Serial로 출력할 문자열이 있는 경우
        buf[pos] = 0;             // 문자열의 끝 표시
        Serial.print(buf);

        pos = 0;                  // 버퍼가 비었음을 표시
        process_it = false;
    }
}
```

스케치 2에서 슬레이브가 수신한 문자는 SPDR (SPI Data Register) 레지스터를 통해 알아낼 수 있다. SPDR 레지스터의 값을 읽으면 SPI 통신을 통해 전달된 데이터가 저장되는 데이터 버퍼에 저장된 값을 읽어올 수 있다.

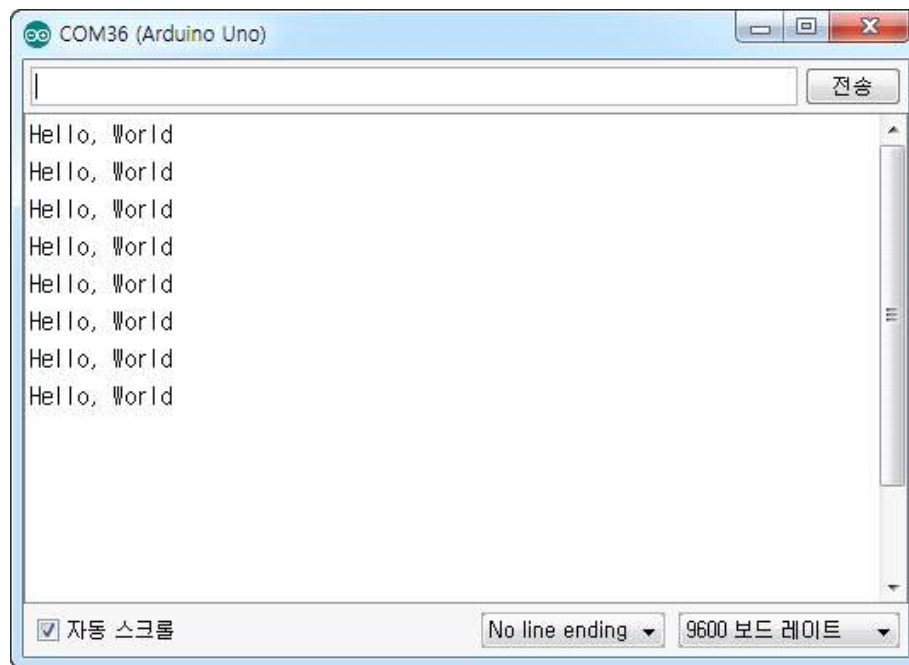


그림 2. 스케치 1과 스케치 2 실행 결과

2. 슬레이브로부터 데이터 수신

SPI는 원형 큐를 사용하므로 슬레이브로부터 데이터를 수신하기 위해서도 슬레이브로 데이터를 송신하여야 한다. 즉, 1바이트의 데이터를 얻기 위해서는 1바이트의 데이터를 보내야만 한다. 슬레이브는 1초 간격으로 0에서 9까지 증가하는 카운터를 가지고 있다고 가정해 보자. 마스터는 시리얼 모니터를 통해 사용자의 입력을 기다리며 문자 'C'가 입력되는 경우 슬레이브로부터 현재 카운터 값을 전달 받아 이를 시리얼 모니터로 출력하도록 해보자. 슬레이브에는 스케치 3을 업로드 한다. 스케치 2에서 슬레이브가 SPDR 레지스터를 통해 수신된 문자를 읽어오는 것과 유사하게 스케치 3에서 마스터로 데이터 전송을 위해 SPDR 레지스터를 사용하고 있다.

스케치 3. 초 단위 카운터 - 슬레이브

```
#include <SPI.h>

byte count;
```

```
void setup (void)
{
    // SPI 통신을 위한 핀들의 입출력 설정
    pinMode(MISO, OUTPUT);
    pinMode(MOSI, INPUT);
    pinMode(SCK, INPUT);
    pinMode(SS, INPUT);

    // 마스터의 전송 속도에 맞추어 통신 속도를 설정한다.
    SPI.setClockDivider(SPI_CLOCK_DIV16);

    // SPI 통신을 위한 레지스터를 설정
    SPCR |= _BV(SPE);                // SPI 활성화
    SPCR &= ~_BV(MSTR);              // Slave 모드 선택
    SPCR |= _BV(SPIE);               // 인터럽트 허용

    count = 0;                       // 카운터 초기화
}

// SPI 통신으로 데이터가 수신될 때 발생하는 인터럽트 처리 루틴
ISR (SPI_STC_vect)
{
    SPDR = count + '0';              // 카운터 값을 ASCII 값으로 전달
}

void loop (void)
{
    count = (count + 1) % 10;         // 카운터 값 증가
    delay(1000);
}
```

마스터에는 스케치 4를 업로드 한다.

스케치 4. 카운터 값 요청 - 마스터

```
#include <SPI.h>

void setup (void)
{
    SPI.begin ();                // SPI 통신 초기화
    digitalWrite(SS, HIGH);      // 슬레이브가 선택되지 않은 상태로 유지

    // 안정적인 전송을 위해 분주비를 높여 전송 속도를 낮춤
    SPI.setClockDivider(SPI_CLOCK_DIV16);

    Serial.begin(9600);
}

void loop (void)
{
    if(Serial.available()){
        char data = Serial.read();    // 데이터 입력 확인
        if(data == 'C'){
            digitalWrite(SS, LOW);     // 슬레이브를 선택한다.
            // 1바이트 데이터 수신을 위해 의미 없는 1바이트 데이터를 전송한다.
            char received = SPI.transfer(0);
            digitalWrite(SS, HIGH);    // 슬레이브 선택을 해제한다.
            Serial.println(received);
        }
    }
}
```

마스터와 연결된 시리얼 모니터를 통해 문자 'C'를 입력하여 슬레이브의 현재 카운터 값이 전송되어 오는지 확인해 보자.

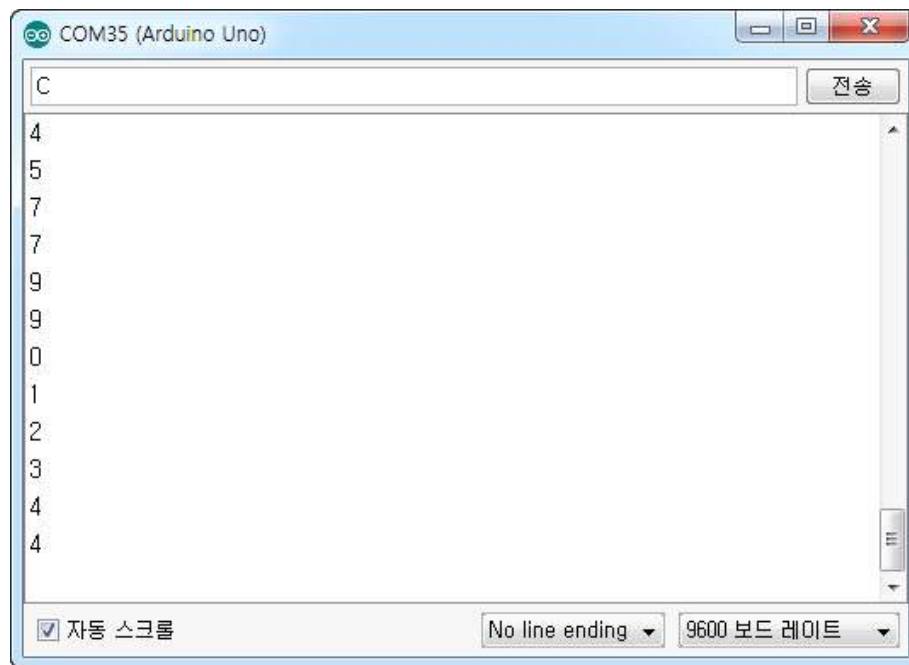


그림 3. 스케치 3과 스케치 4 실행 결과