# 아두이노 통신

Professor H.J. Park, Dept. of Mechanical System Design, Seoul National University of Science and Technology.

An Unmanned aerial vehicle (UAV) is a Unmanned Aerial Vehicle. UAVs include both autonomous (means they can do it alone) drones and remotely piloted vehicles (RPVs).

# CONTENTS


# 아두이노 통신

# Arduino 통신

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- 3 종류의 시리얼 통신
  - UART (0~3), SPI, TWI(I2C)
- **SPI**
  - Full-duplex, Three-wire Synchronous Data Transfer:
    - 양방향 동기식.
  - Master or Slave Operation:
  - Up to 20MHz data speed
- **TWI**
  - 7-bit Address:
    - 총 128 Slave.
  - Half-duplex, two-wire, Asyncronous
  - Up to 400kHz Data Transfer Speed:

# Arduino 통신

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- 양방향 통신은 다음 2가지 경우로 나뉜다.
  - Full duplex: 송신/수신을 동시에 가능
    - USART, SPI
  - Half duplex: 송신/수신을 번갈아 가면서 수행
    - I2C

| BUS | 동기방식 | 1:n | 양방향 | 1:1인경우 최소 선수 |
|---|---|---|---|---|
| UART | baud rate | x | Full | 3<br>TX, RX |
| I2C | clock line<br>(SCL) | ok<br>(address) | Half | 3<br>SDA,SCL |
| SPI | clock line<br>(SCK) | ok<br>(chip select) | Full | 4<br>MISO,MOSI,SCK |

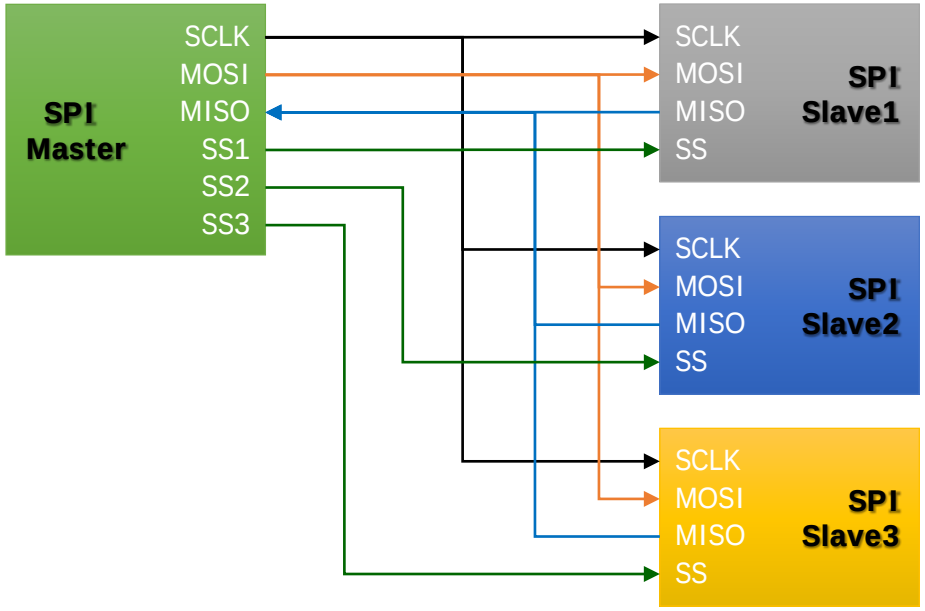

# SPI 통신

# SPI 개요

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- SPI : Serial Peripheral Interface
  - MCU와 다른 장치 (PC, MCU)들과 정보교환에 유용
  - 최소 4선이 필요

| 신호 | 의미 | 방향 | 비고 |
|---|---|---|---|
| **SCLK** | Serial Clock | From Master | |
| **MOSI** | Master Output, Slave Input | | |
| **SS** | Slave Select | | Low active |
| **MISO** | Master Input, Slave Output | From Slave | |



# SPI 장단점



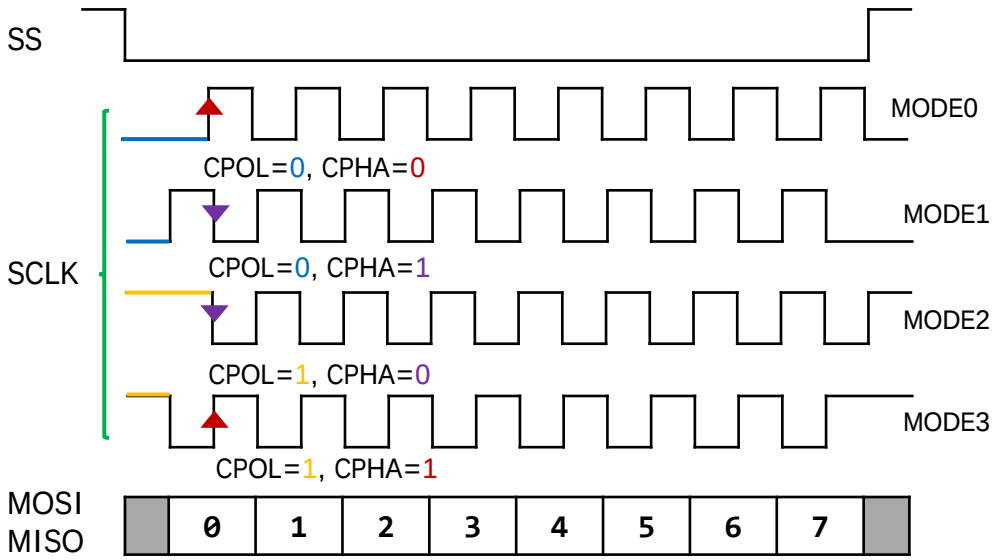
- 장점
  - Full duplex communication
  - Lower power requirements , Higher throughput than I²C or SMBus
  - Push-pull drivers provide good signal integrity and high speed
  - Lower power requirements , Higher throughput than I²C or SMBus
- 단점
  - Requires more pins and wires (min. 4pins)

# SPI 개요

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

## SPI 동작

- 모드: Clock의 극성과 데이터 읽기 위상 결정
  - CPOL : Clock이 어떤 상태로 시작하느냐를 결정
  - CPHA : 데이터 읽는 엣지 시점 정의
    - = 0 : CPOL의 반대 상태로
    - = 1 : CPOL 상태로

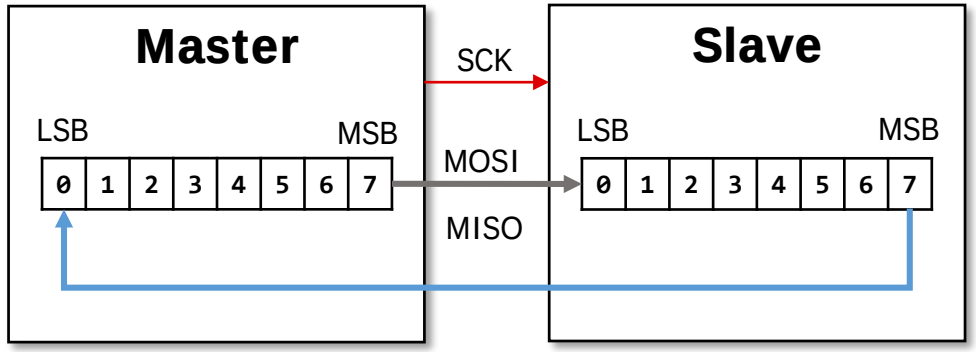



| Mode | Clock Polarity (CPOL) | Clock Phase (CPHA) | Data Capture |
|---|---|---|---|
| SPI_MODE0 | 0 | 0 | Rising |
| SPI_MODE1 | 0 | 1 | Falling |
| SPI_MODE2 | 1 | 0 | Falling |
| SPI_MODE3 | 1 | 1 | Rising |

# SPI 개요

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- 데이터 전송 bit 순서
  - MSBFIRST: most-significant bit first



# Arduino에서의 SPI

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

## SPI 객체 사용 (SPI.h)

- 라이브러리 사용

```
#include <SPI.h>
```

  - 라이브러리 include

- SPI 시작

```
SPI.begin()
```

  - Initializes the SPI bus by setting SCK, MOSI, and SS to outputs, pulling SCK and MOSI low, and SS high.

- SPI 설정

```
SPI.beginTransaction(SPISettings(clock, bitOrder, dataMode))
```

  - *clock*: clock speed 1~20MHz
  - *bitOrder*: MSBFIRST or LSBFIRST
  - *dataMode*: `SPI_MODE0, SPI_MODE1, SPI_MODE2, SPI_MODE3`

- SPI 전송 및 수신

```
ret = SPI.transfer(val)
```

  - *val*: the byte to send out over the bus
  - *ret*: received data

# Arduino 에서의 SPI

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- SPI 예제
  - 속도, 비트순서, 모드 설정

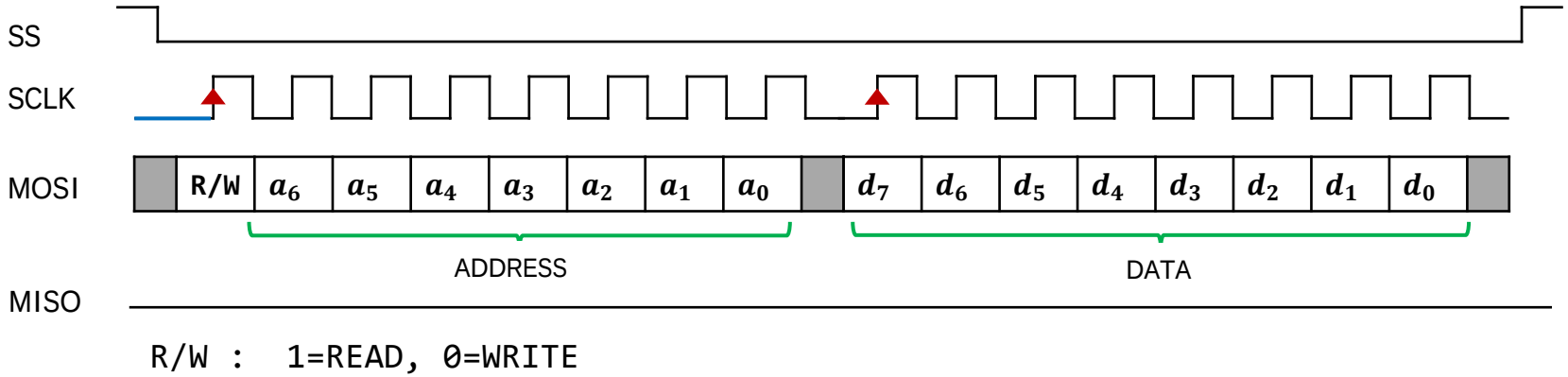
```
#include <SPI.h>
#define ChipSelPin 53;
...
  SPI.begin();
  SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0));
  pinMode(ChipSelPin, OUTPUT);
```

# Arduino 에서의 SPI

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

## SPI 예제

- Address에 data 쓰기




R/W : 1=READ, 0=WRITE

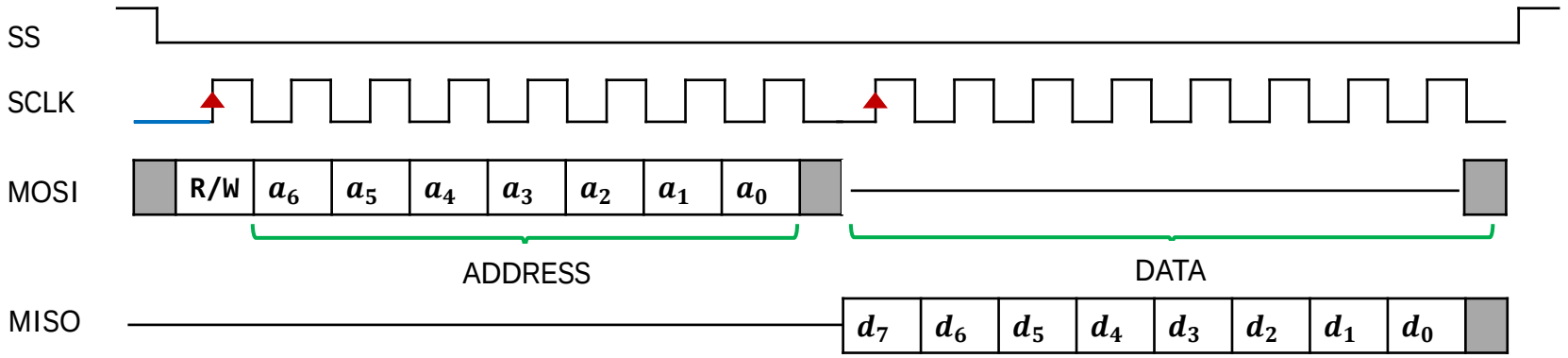
- 코드
  - Chip select → LOW
  - .transfer : address (8bits)
  - .transfer : data
  - Chip select → HIGH

```
void SPIwrite(byte reg, byte data) {
  uint8_t dump;
  digitalWrite(ChipSelPin,LOW);
  dump=SPI.transfer(reg);
  dump=SPI.transfer(data);
  digitalWrite(ChipSelPin,HIGH);
}
```

# Arduino 에서의 SPI

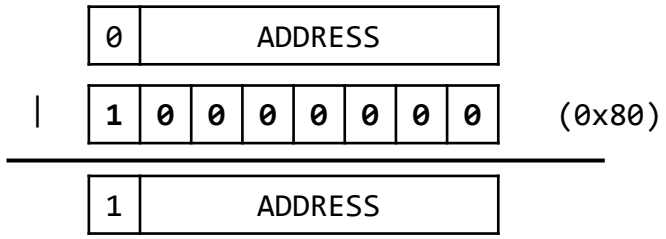
Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- SPI 예제
  - Address에서 data 읽기




R/W : 1=READ, 0=WRITE

  - 코드
    - read! : addr = ADDRESS | 0x80;
    - Chip select → LOW
    - .transfer : address (8bits)
    - .transfer : 0 → data return !
    - Chip select → HIGH



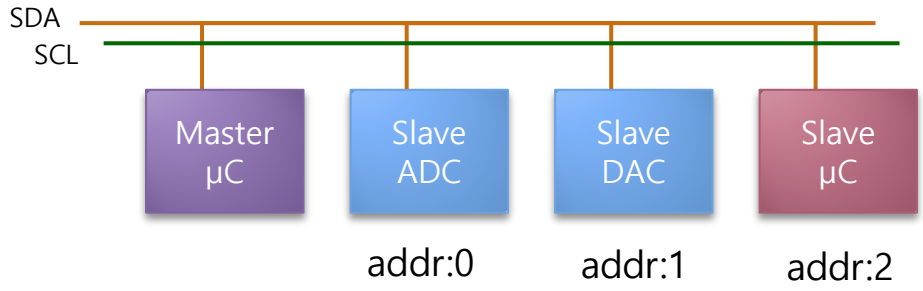

```
uint8_t SPIread(byte reg,int ChipSelPin) {
  uint8_t dump;
  uint8_t return_value;
  uint8_t addr=reg|0x80;
  digitalWrite(ChipSelPin,LOW);
  dump=SPI.transfer(addr);
  return_value=SPI.transfer(0x00);
  digitalWrite(ChipSelPin,HIGH);
  return(return_value);
}
```


# I2C 통신

# I2C 통신

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- I2C는 1982년 Philips가 개발하였고 TWI라는 별칭도 있다.
  - 2선만으로 복수개의 장치와 통신이 가능하다는 것이 장점이다.
  - 초기에는 400kHz 가 최대 속도였으나 최근에는 고속 모드를 지원한다.
  - 변종으로 Intel의 SMBus가 있다.
- 주요 특징
  - SDA, SCL 라인을 2k~10kΩ 정도로 Pull-up 시켜 사용.
  - master, slave 모두 Open drain 회로로  0으로 만들 수 있다.



# Arduino에서의 I2C

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

- 라이브러리 사용
  - #include <Wire.h>
- 통신의 초기화
  - Master/Slave, Slave 주소

Wire.***begin(address);***

- *address* : 생략 → Master mode, Slave 주소 → Slave mode

# Arduino에서의 I2C



## 통신시작

- Slave주소 지정

```
Wire.beginTransmission(address);
```

- *address* : 마스터가 통신하고자하는 Slave 주소 (0~127)

# Arduino에서의 I2C



## 저장된 데이터 전송 종료

- master mode에서만 사용한다.

*s* = Wire.***endTransmission(stop);***

- *stop* : 정지신호 생성 여부, TRUE(생략 시 기본값): 생성, FALSE: 계속
- *s* : 전송성공여부: 0: 성공, 1:데이터 길이가 버퍼를 초과, 2: 해당 주소 대답 안함, 3, 데이터 전송 실패, 4: 그 외 오류

# Arduino에서의 I2C



## 데이터 전송 요청

- Slave에게 데이터 전송을 요청한다.

*b*=Wire.***requestFrom(address, quantity, stop);***

- address : 통신하고자하는 Slave 주소 (0~127)
- quantity: 요청하는 데이터의 바이트 수
- stop : 정지신호 생성 여부, TRUE(생략시 기본값): 생성, FALSE: 계속
- b : 요청 후 버퍼에 수신된 바이트 수

# Arduino에서의 I2C

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

## 데이터 쓰기

Wire.***write(value);***

- value : 전송하고자 하는 byte형 데이터

Wire.***write(string);***

- string : 전송하고자 하는 문자열 데이터

Wire.***write(dataArray, length);***

- dataArray : 전송하고자 하는 byte형 배열
- length: dataArray의 길이 (바이트수)

# Arduino에서의 I2C



## 데이터 읽기

- Wire.requestFrom 으로 요청한 후 수신버퍼에서 1바이트 읽어 들인다.

*byte*=Wire.***read();***

- *byte* : 수신된 1바이트

## 수신버퍼 확인

- 수신 버퍼에 수신된 바이트 수 확인

*nbye*=Wire.***available();***

- *nbyte* : 버퍼에 수신된 바이트 수

# Arduino 에서의 I2C

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

## I2C 예제

- register 에 data 쓰기
  - .beginTransmission(ADR)
  - .write (reg)
  - .write (data)
  - . endTransmission()

```
void writeRegister8(uint8_t reg, uint8_t value){
    Wire.beginTransmission(CHIP_ADDRESS);
    Wire.write(reg);
    Wire.write(value);
    Wire.endTransmission();
}
```

# Arduino 에서의 I2C

Dept. of Mechanical System Design, Seoul National University of Science and Technology.

## I2C 예제

- register에서 data 읽기
  - .beginTransmission(ADR)
  - .write (reg)
  - .endTransmission()
  - .beginTransmission(ADR)
  - .requestFrom(ADR,1)
  - wait .available()
  - .read(data) → value
  - . endTransmission()

```
uint8_t readRegister8(uint8_t reg) {
    uint8_t value;
    Wire.beginTransmission(CHIP_ADDRESS);
    Wire.write(reg);
    Wire.endTransmission();
    Wire.beginTransmission(CHIP_ADDRESS);
    Wire.requestFrom(CHIP_ADDRESS, 1);
    while(!Wire.available()) {};
    value = Wire.read();
    Wire.endTransmission();
    return value;
}
```

# **THANK** YOU

An Unmanned aerial vehicle (UAV) is a Unmanned Aerial Vehicle. UAVs include both autonomous (means they can do it alone) drones and remotely piloted vehicles (RPVs).