Internet of Things
class 6

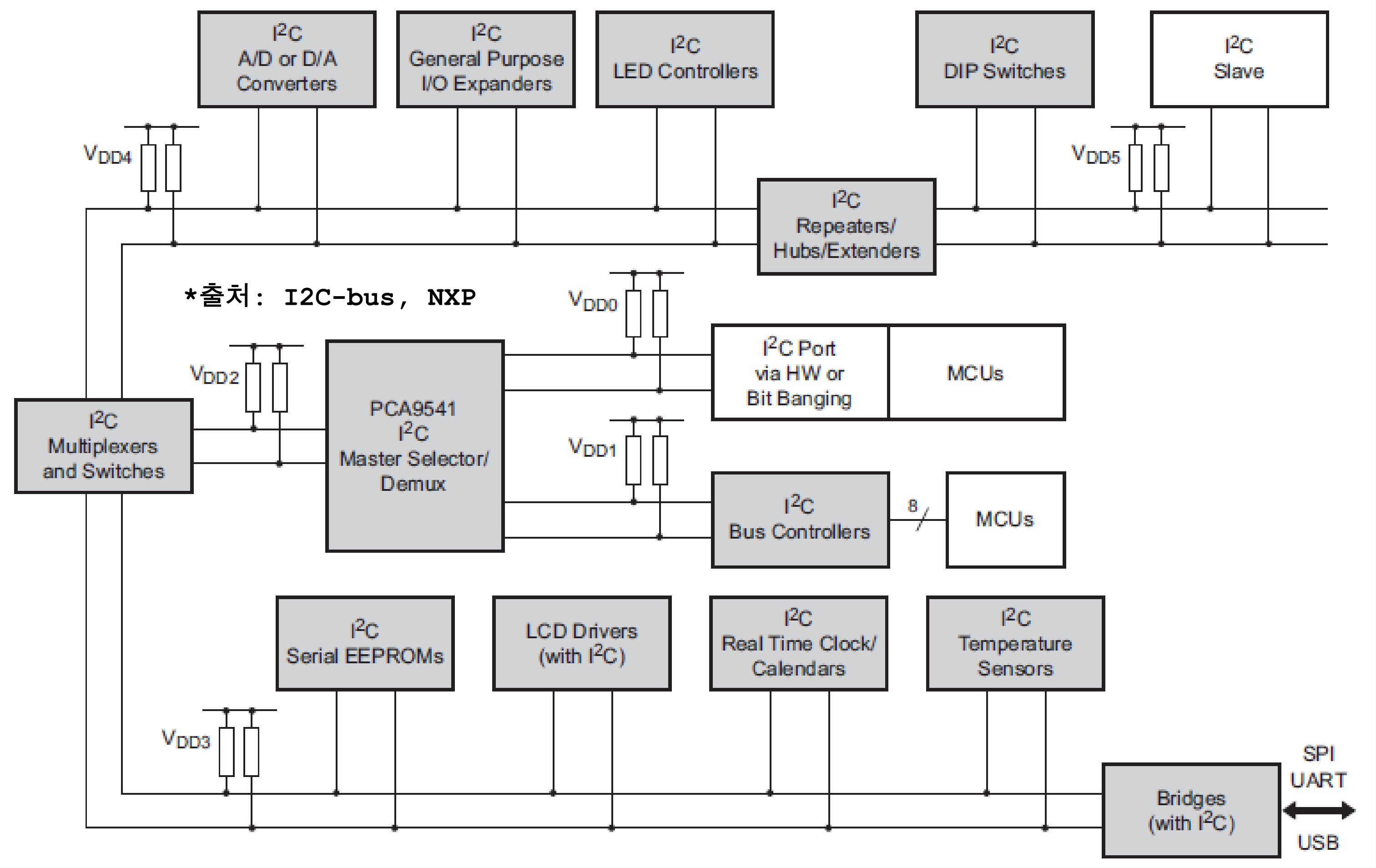
I^2C, SPI, EEPROM

Short-distance Communication(1): I²C

- Inter-Integrated Circuit (I²C)
  - Multi-master, multi-slave, packet switched, single-ended, serial computer bus invented by Philips Semiconductor(NXP)
  - Typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance, intra-board communication
  - I²C uses only two bidirectional open-drain lines, Serial Data Line (SDA) and Serial Clock Line (SCL)
  - The I2C reference design has a 7-bit or a 10-bit (depending on the device used) address space
  - Common I²C bus speeds are the 100 kbit/s *standard mode* and the 10 kbit/s *low-speed mode*, but arbitrarily low clock frequencies are also allowed
    - Fast Mode: 400 kbit/s
    - Fast Mode Plus(Fm+): 1 Mbit/s
    - High Speed Mode: 3.4 Mbit/s

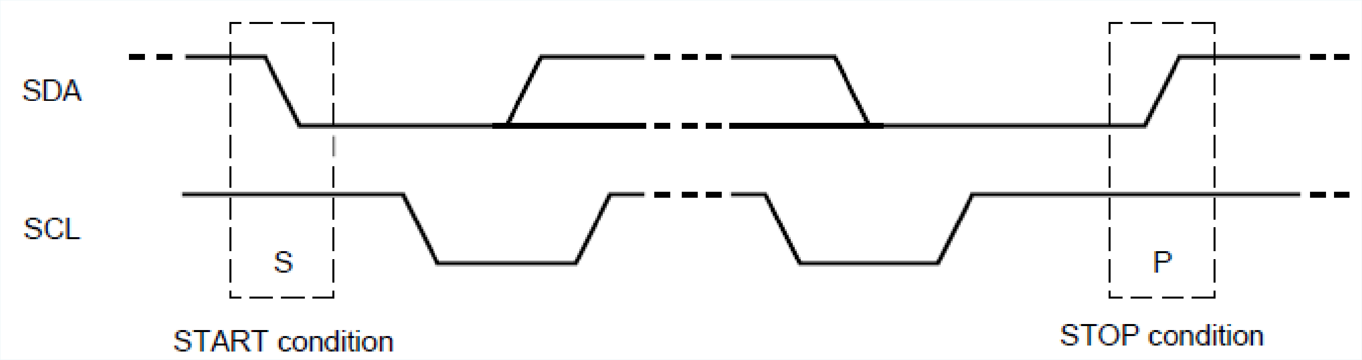
I2C (I²C)

- Example of I2C Bus Applications

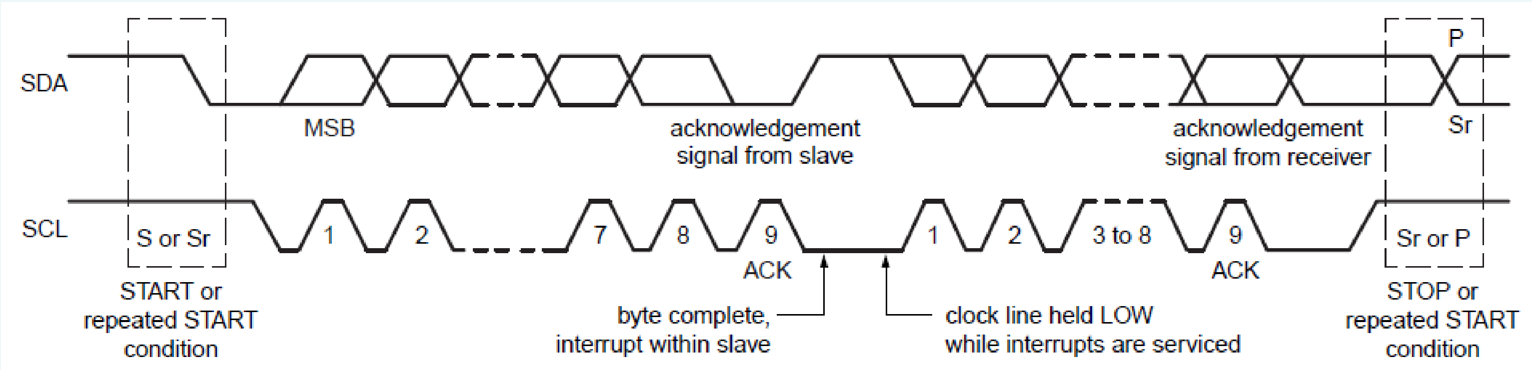


I2C

- I2C Communication
  - START and STOP conditions
    - All transactions begin with a START (S) and are terminated by a STOP (P)
    - HIGH to LOW transition on the SDA line while SCL is HIGH defines a START condition
    - LOW to HIGH transition on the SDA line while SCL is HIGH defines a STOP condition
    - START and STOP conditions are always generated by the master

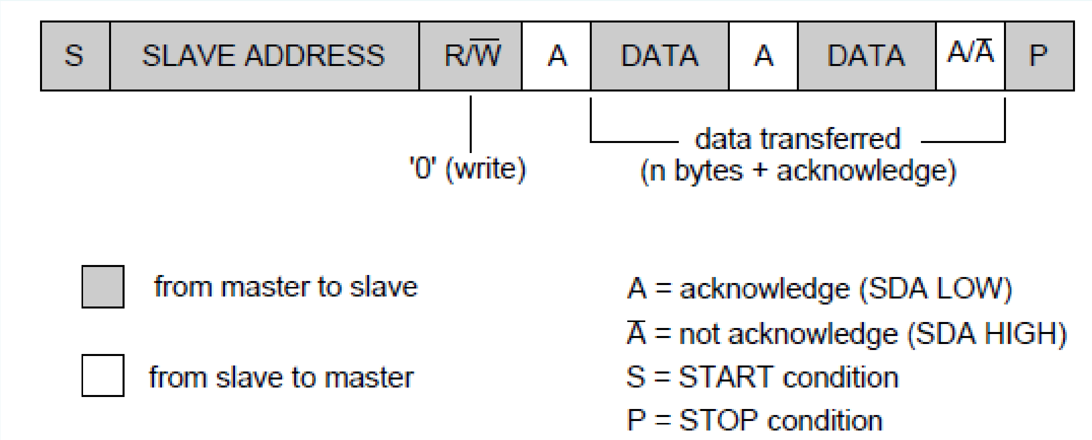


- I2C Communication
  - Byte Format
    - Every byte put on the SDA line must be eight bits long
    - Each byte must be followed by an ACK bit
    - Data is transferred with the MSB first
    - If a slave cannot receive or transmit another complete byte of data until it has performed some other function:
      - It can hold the clock line SCL LOW to force the master into a wait state
      - Data transfer then continues when the slave is ready for another byte of data and releases clock line SCL

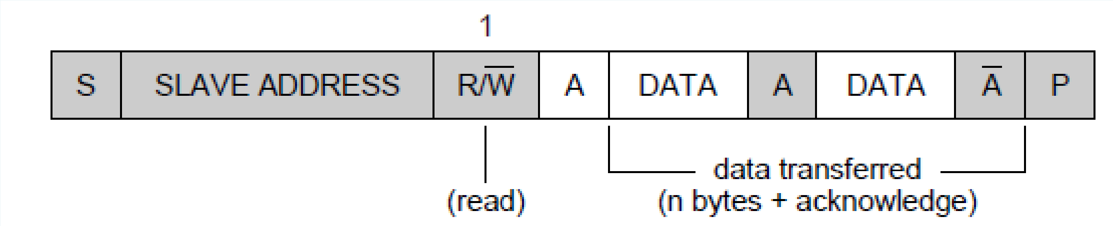


I2C

- I2C Communication
  - Byte Format
    - A master-transmitter addressing a slave receiver with a 7-bit addr

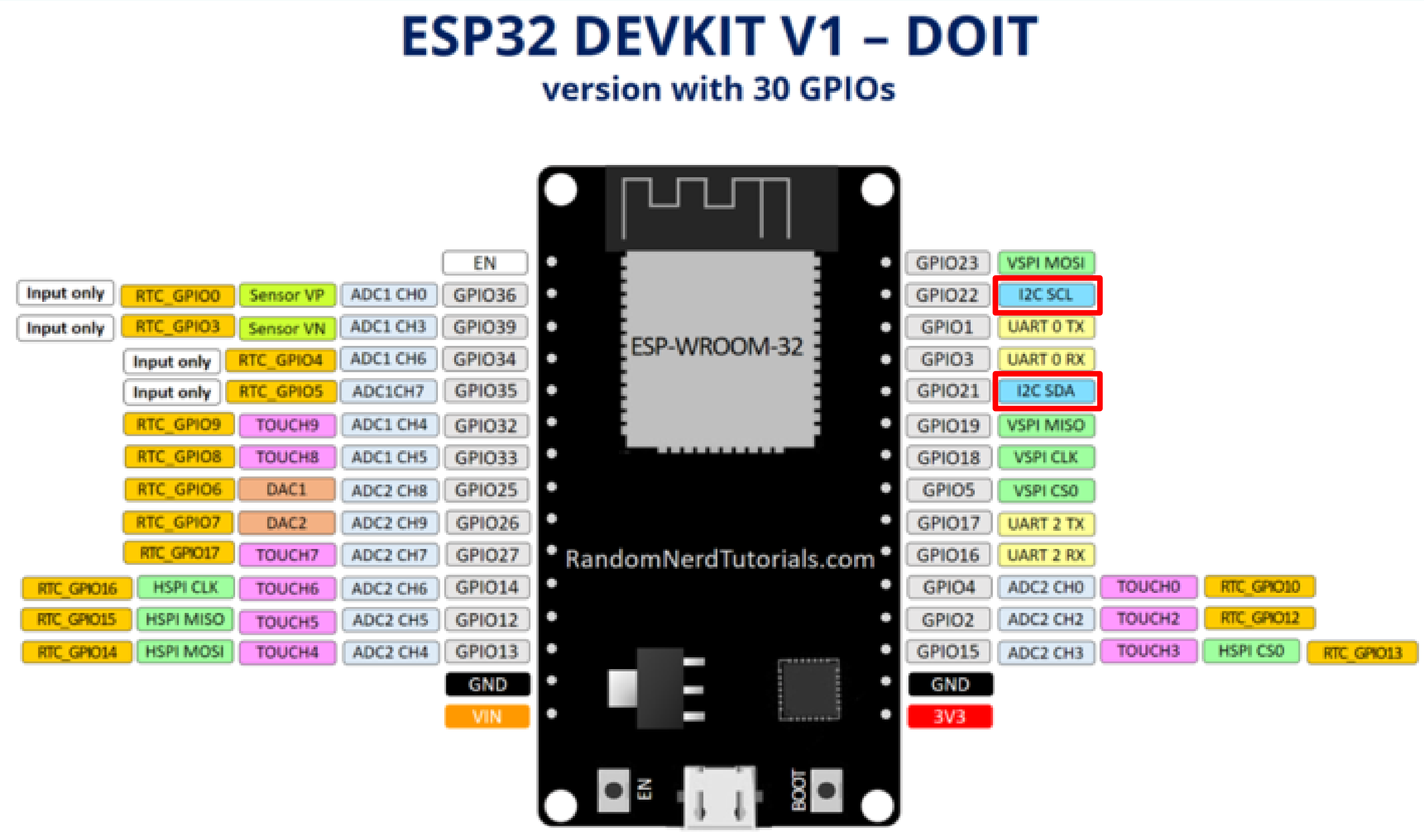


    - A master reads a slave immediately after the first byte



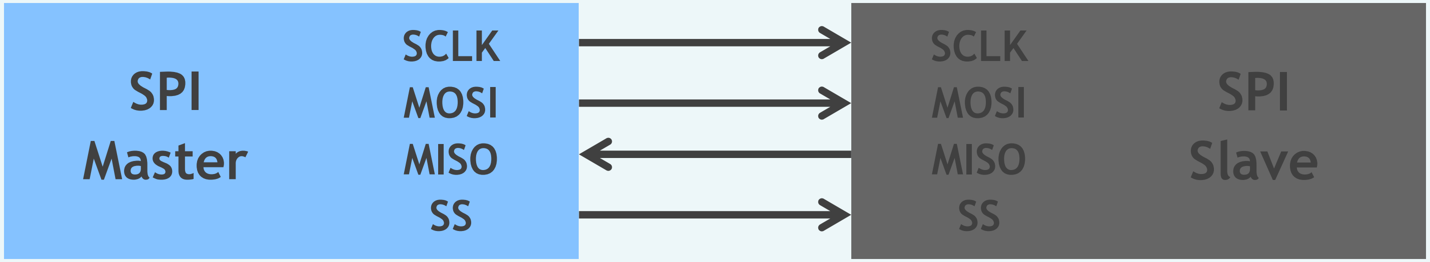
ESP32 I2C Interfaces

- ESP32 I2C Interface
  - I2C0: GPIO22 (SCL), GPIO21 (SDA)



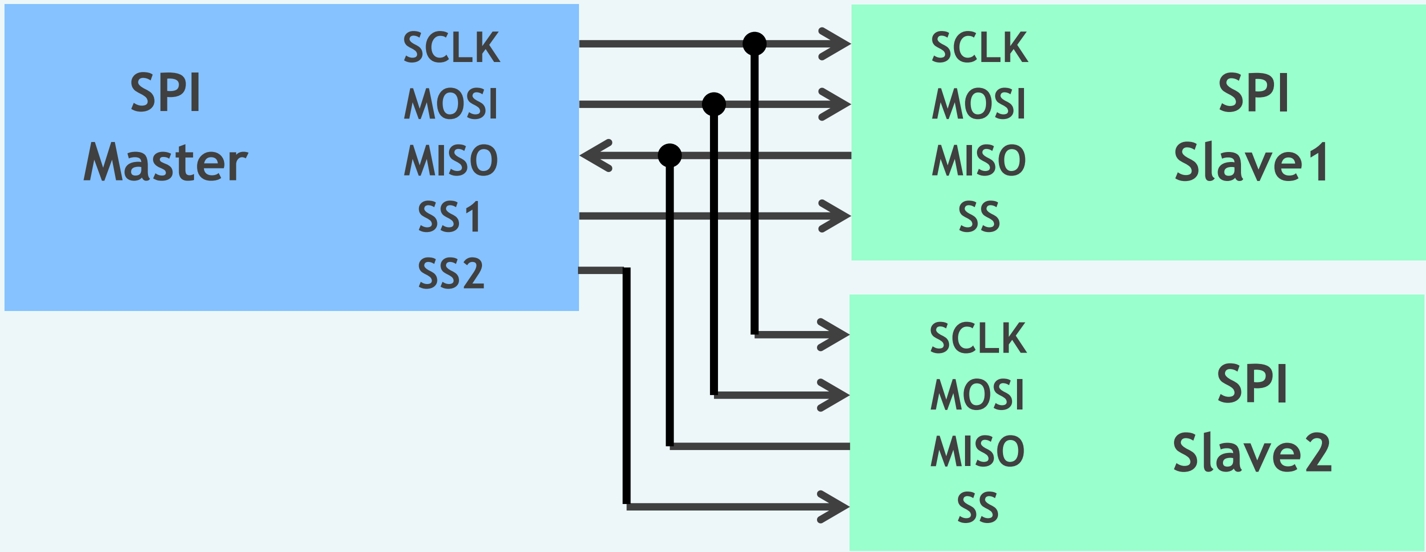
Short-distance Communication(2): SPI

- Serial Peripheral Interface (SPI)
  - Synchronous serial communication interface specification used for short distance communication
  - The interface was developed by Motorola in the late 1980s and has become a *de facto* standard
  - SPI devices communicate in full duplex mode using a master-slave architecture with a single master
  - The master device originates the frame for reading and writing
  - Multiple slave devices are supported through selection with individual slave select (SS) lines
  - SPI bus speeds are 10 Mbit/s ~ 20Mbit/s



Short-distance Communication(2): SPI

-



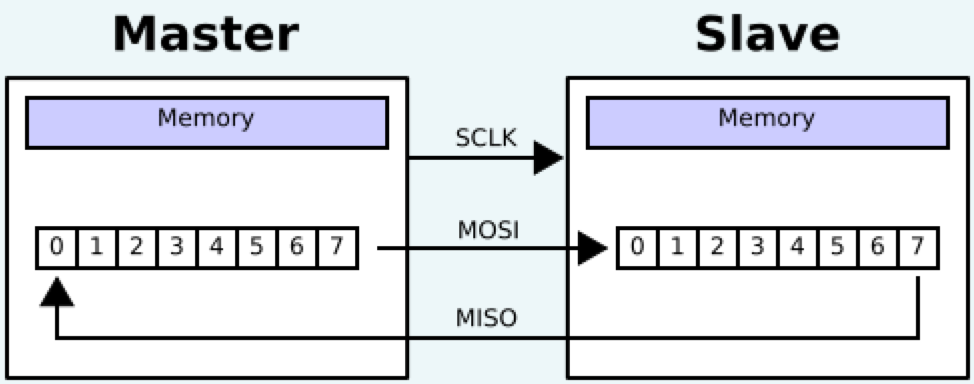
C.B.Choi gen1223@kau.ac.kr

SPI

- Serial Paralle Interface (SPI)
  - Four logic signals for SPI Bus
    - SCLK: Serial Clock (output from master)
    - MOSI: Master Output Slave Input (data output from master)
    - MISO: Master Input Slave Output (data output from slave)
    - SS(CS): Slave(Chip) Select (often active low, output from master)
  - Alternative names:
    - SCLK: SCK, CLK
    - MOSI: SIMO, SDI, DI, DIN, SI
    - MISO: SOMI, SDO, DO, DOUT, SO
    - SS: nCS, CS, CSB, CSN, nSS, STE

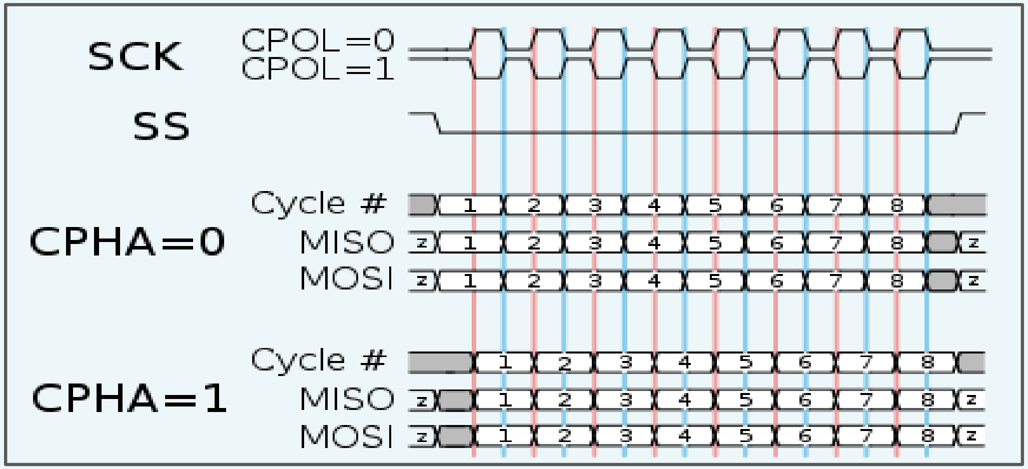
SPI

- SPI Data Transmission
  - To begin communication
    - Bus master configures the clock, using a frequency supported by the slave device (a few MHz)
    - The master then selects the slave device with a logic level 0 on the select line
    - During each SPI clock cycle, a full duplex data transmission occur
    - The master sends a bit on the MOSI line and the slave reads it, while the slave sends a bit on the MISO line and the master reads it



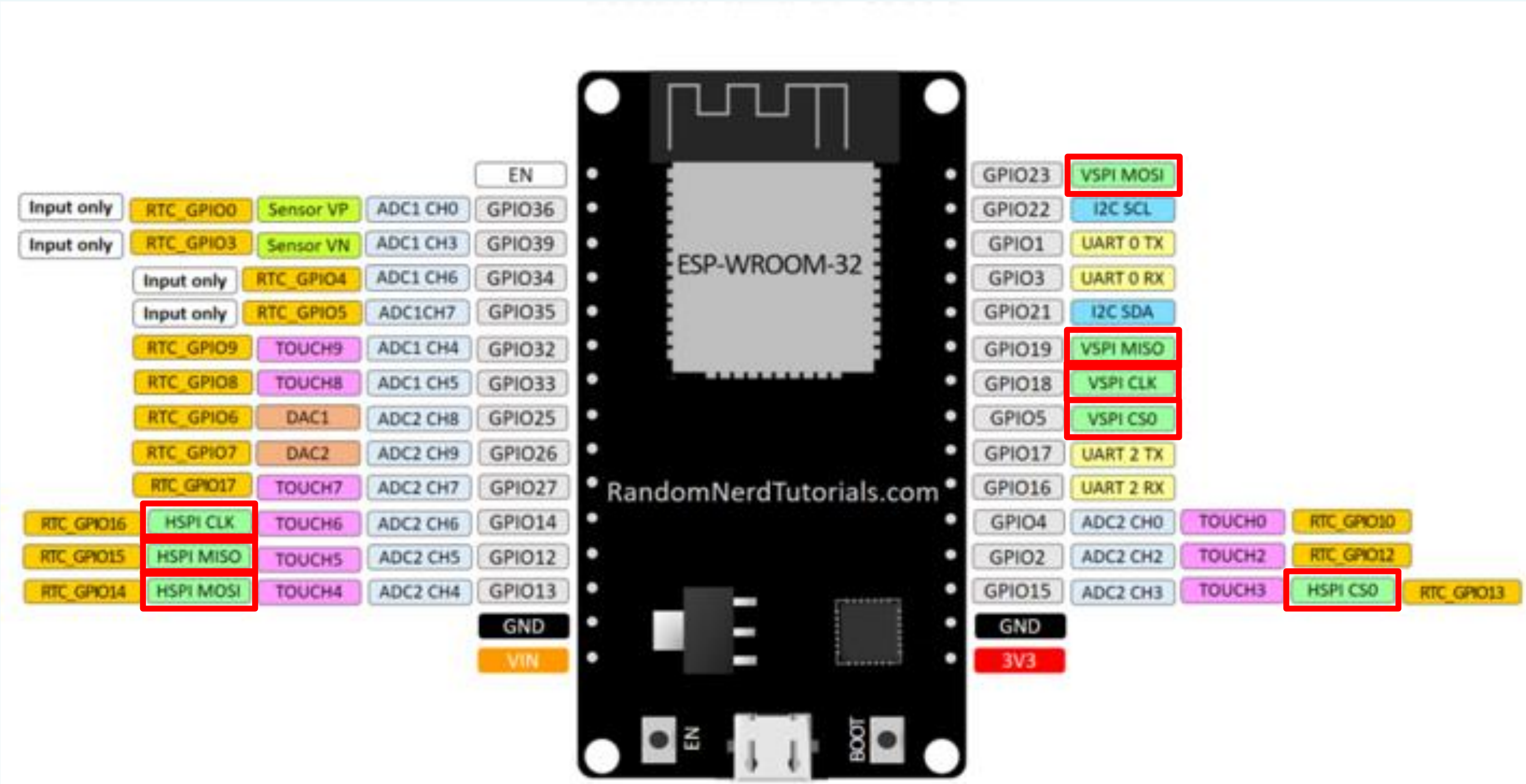
SPI

- Clock Polarity and Phase
  - Master must also configure the clock polarity(CPOL) and phase(CPHA)
    - CPOL=0: base value of the clock is zero (idle state 0, active state 1)
      - » CPHA=0, data are captured and output on the clock's rising edge (low→high transition)
      - » CPHA=1, data are captured and output on the clock's falling edge
    - CPOL=1: base value of the clock is one (idle state 1, active state 0)
      - » CPHA=0, data are captured and output on clock's falling edge
      - » CPHA=1, data are captured and output on clock's rising edge



ESP32 SPI Interfaces

- ESP32 SPI Interface
  - HSPI: GPIO14 (SCL), GPIO12 (MISO), GPIO13 (MOSI), GPIO15 (SS)
  - VSPI: GPIO18 (SCL), GPIO19 (MISO), GPIO23 (MOSI), GPIO5 (SS)

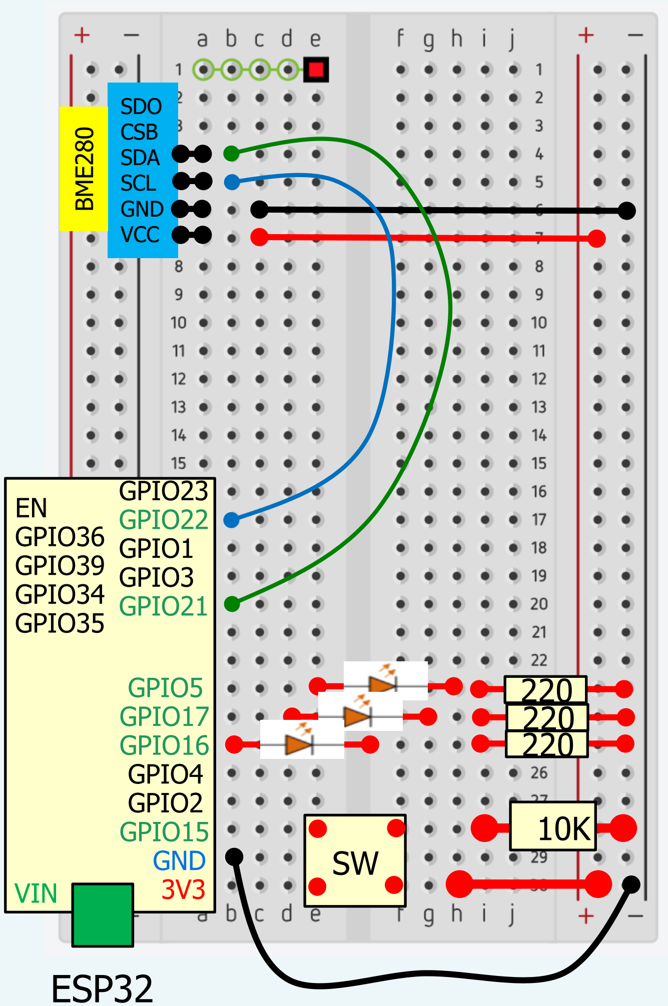


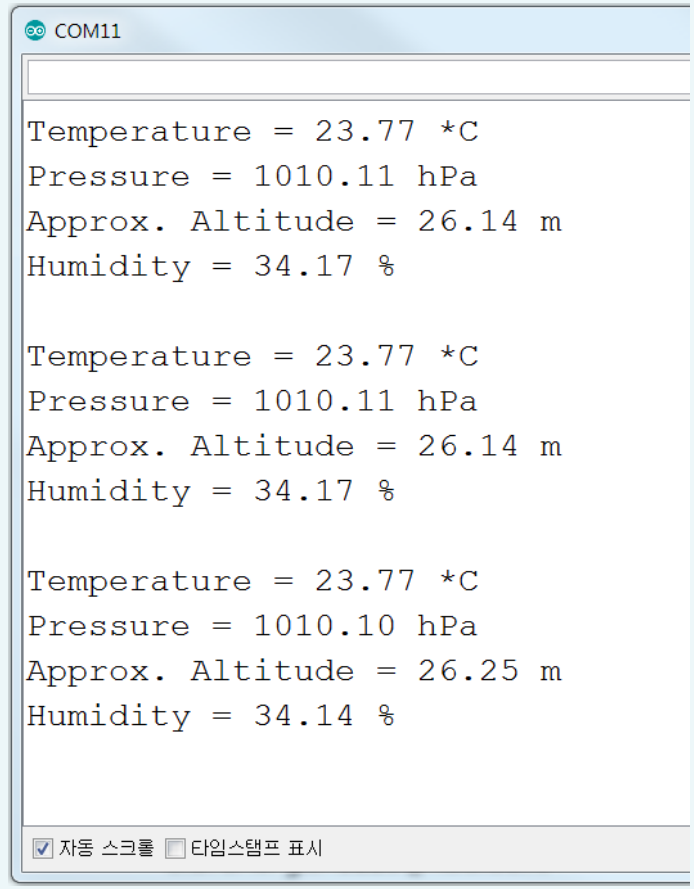
Example: BME280 Sensor Module

- BME280 reads Temperature, Humidity, Pressure
  - Can communicate using either SPI or I2C communication
  - Use the following pins for I2C communication protocol:
    - SCL – I2C Clock
    - SDA – I2C Data
  - Use the following pins for SPI communication protocol:
    - SCL – SPI Clock (CLK)
    - SDO – MISO
    - SDA – MOSI
    - CSB – Chip Select (SS)

BME280 with I2C

<Task06-1> ** BME280의 SDA, SCL을 GPIO21, 22에 연결 **

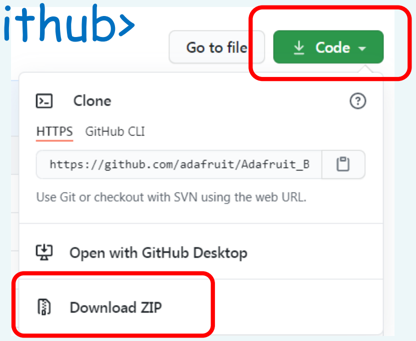




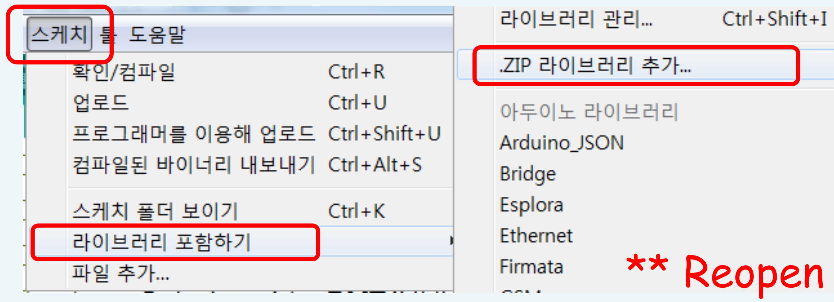
Install New Libraries (BME280)

- Install BME280 Library: Adafruit_BME280
  - url.. https://github.com/adafruit/Adafruit_BME280_Library

<Github>



<Arduino IDE>



** Reopen Arduino IDE

Adafruit_BME280_Library-master.zip

- Install Adafruit_Sensor library
  - url.. https://github.com/adafruit/Adafruit_Sensor
  - Download zip: Adafruit_Sensor-master.zip
  - 위와 동일하게 실행
  - Arduino IDE 재실행

BME280 with I2C

<Task06-1>

```
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>
#define SEALEVELPRESSURE_HPA (1013.25)

Adafruit_BME280 bme; // I2C
int delayTime;

void setup() {
    bool status;
    Serial.begin(115200);
    Serial.println(F("BME280 test"));
// default settings
    status = bme.begin(0x76);       // bme280 I2C address = 0x76
    if (!status) {
        Serial.println("Could not find a valid BME280 sensor, check wiring, address, sensor ID!");
        Serial.print("SensorID was: 0x"); Serial.println(bme.sensorID(),16);
        Serial.print("       ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
        Serial.print("       ID of 0x56-0x58 represents a BMP 280,\n");
        Serial.print("       ID of 0x60 represents a BME 280.\n");
        Serial.print("       ID of 0x61 represents a BME 680.\n");
        while (1) delay(10);
    }
```

BME280 with I2C

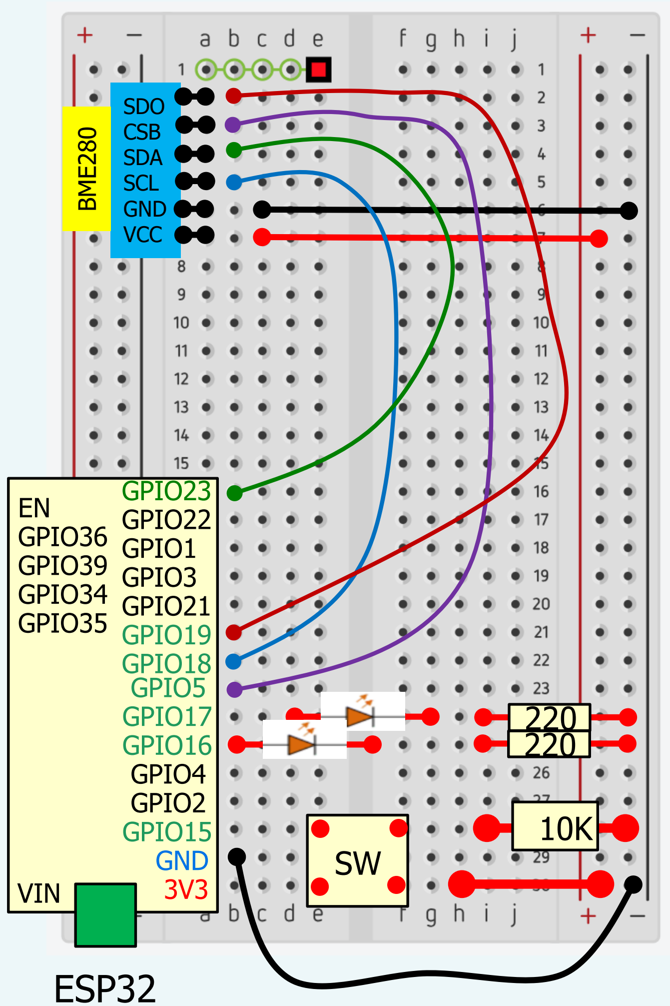
<Task06-1>

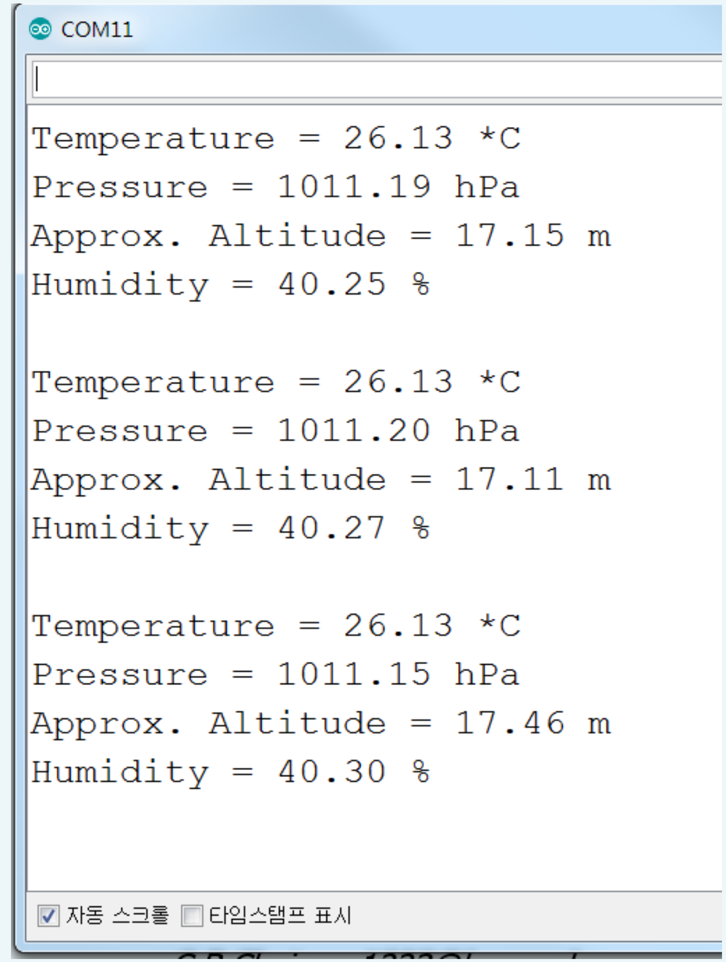
```
    Serial.println("-- Default Test --");
    delayTime = 1000;
    Serial.println();
}

void loop() {
    printValues();
    delay(delayTime);
}
```

```
void printValues() {
    Serial.print("Temperature = ");
    Serial.print(bme.readTemperature());
    Serial.println(" *C");

    Serial.print("Pressure = ");
    Serial.print(bme.readPressure() / 100.0F);
    Serial.println(" hPa");

    Serial.print("Approx. Altitude = ");
    Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
    Serial.println(" m");

    Serial.print("Humidity = ");
    Serial.print(bme.readHumidity());
    Serial.println(" %");
    Serial.println();
}
```

BME280 with SPI

<Task06-2> ** BME280의 SDO, CSB, SDA, SCL을 GPIO19, 5, 23, 18에 연결 **





BME280 with SPI

<Task06-2>

```
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

/* sw spi
#define BME_SCK 13
#define BME_MISO 12
#define BME_MOSI 11 */
#define BME_CS  5       // cs for esp32 vspi

#define SEALEVELPRESSURE_HPA (1013.25)

//Adafruit_BME280 bme; // I2C
Adafruit_BME280 bme(BME_CS); // hardware SPI
//Adafruit_BME280 bme(BME_CS, BME_MOSI, BME_MISO, BME_SCK); // software SPI

int delayTime;
```

BME280 with SPI

<Task06-2>

```
void setup() {
    Serial.begin(115200);
    Serial.println(F("BME280 test"));

    bool status;

    // default settings
    status = bme.begin();
    if (!status) {
        Serial.println("Could not find a valid BME280 sensor, check wiring, address, sensor ID!");
        Serial.print("SensorID was: 0x"); Serial.println(bme.sensorID(),16);
        Serial.print("        ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");
        Serial.print("        ID of 0x56-0x58 represents a BMP 280,\n");
        Serial.print("        ID of 0x60 represents a BME 280.\n");
        Serial.print("        ID of 0x61 represents a BME 680.\n");
        while (1) delay(10);
    }

    Serial.println("-- Default Test --");
    delayTime = 1000;

    Serial.println();
}
```

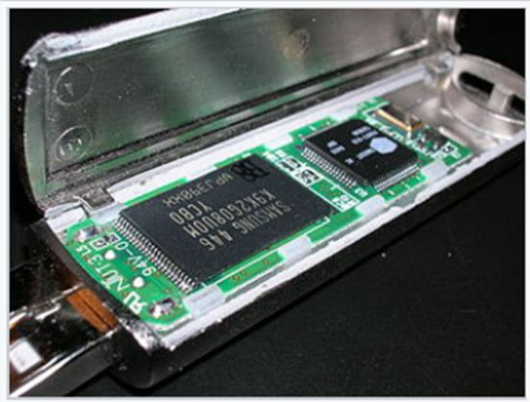
BME280 with SPI

<Task06-2>

```
void loop() {
    printValues();
    delay(delayTime);
}

void printValues() {
    Serial.print("Temperature = ");
    Serial.print(bme.readTemperature());
    Serial.println(" *C");

    Serial.print("Pressure = ");
    Serial.print(bme.readPressure() / 100.0F);
    Serial.println(" hPa");

    Serial.print("Approx. Altitude = ");
    Serial.print(bme.readAltitude(SEALEVELPRESSURE_HPA));
    Serial.println(" m");

    Serial.print("Humidity = ");
    Serial.print(bme.readHumidity());
    Serial.println(" %");
    Serial.println();
}
```

Flash Memory - Store Permanent Data

- Saving data in the flash memory is useful to:
  - Remember the last state of a variable
  - Save settings
  - Any type of data needed to save
- Can read flash memory as many times as you want
- But most devices are designed for:
  - about 100,000 to 1,000,000 write operations



Flash Memory - Store Permanent Data

- Flash memory
  - an electronic (solid-state)
    non-volatile computer memory storage medium

  Block erasure:
  - Sets all bits in the block to 1
  - By erasing the entire block (ex. 64KB)
  - Bit 1 can be set to 0 anytime
  - A finite number of erase cycles requires Flash File System (FFS)

Flash Memory - Store Permanent Data

- EEPROM library for ESP32
  - Similar to the Arduino library
- Up to 512 bytes in the flash memory
  - A value (0~255) in 512 different addresses
- Write

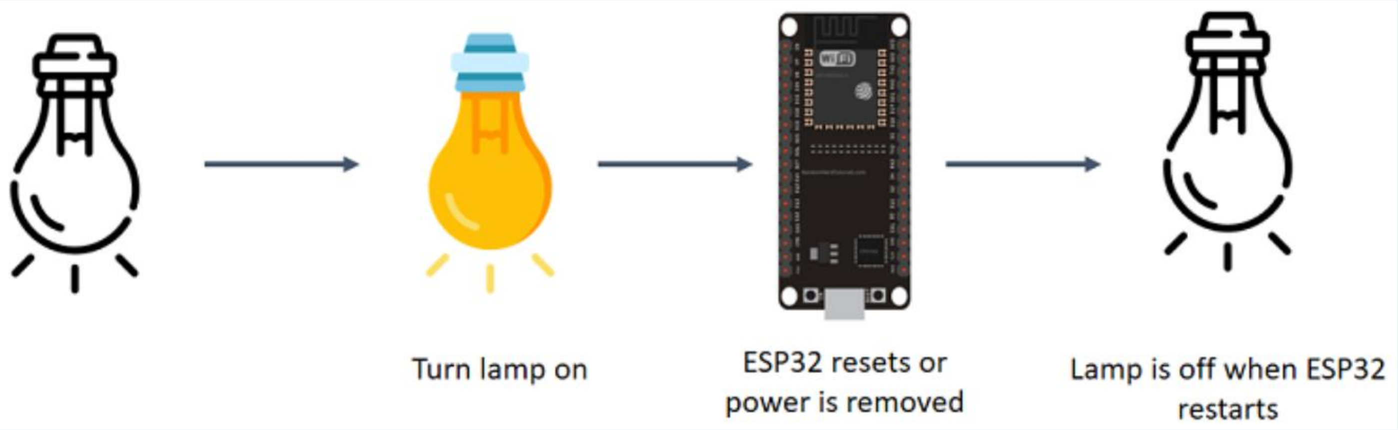
```
EEPROM.write(address, value);
EEPROM.commit();
```

- Read

```
EEPROM.read(address);
```

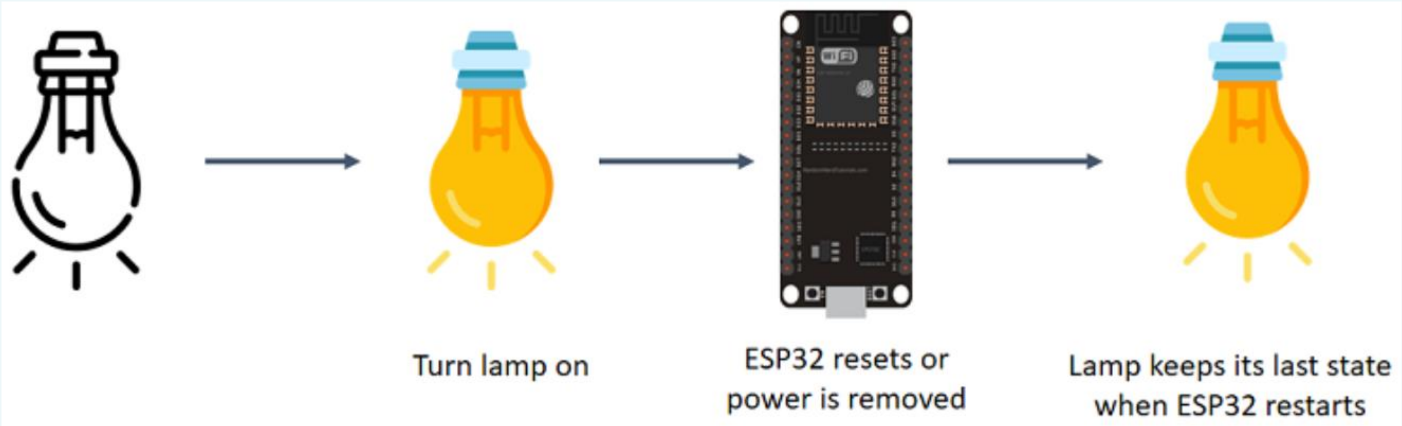
Remember Last GPIO State

1) You're controlling a lamp with the ESP32
2) You set your lamp to turn on
3) The ESP32 suddenly loses power
4) When the power comes back on,
   the lamp stays off
   – because it doesn't keep its last state



Remember Last GPIO State

- You don't want this to happen
  - Remember what was happening before losing power and return to the last state
- To solve this problem,
  - Save the lamp's state in the flash memory
  - Then, you just need to add a condition at the beginning of your sketch to check the last lamp state,
  - And turn the lamp on or off accordingly

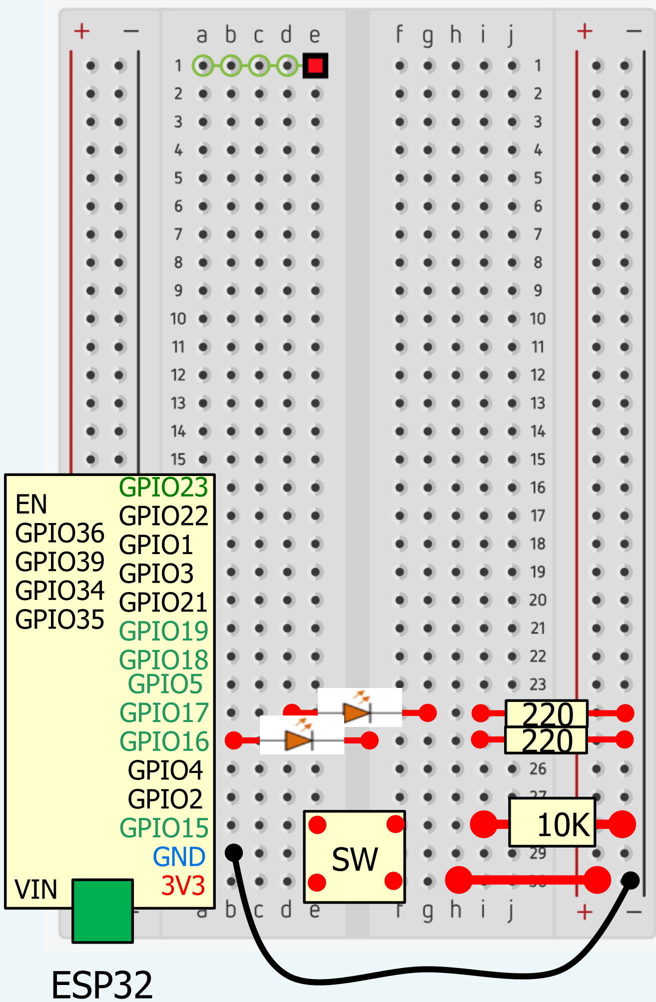


Remember Last GPIO State

<Task06-3> ** Remember LED State **

- Press the pushbutton to turn the LED on and off.
- Reset or
- Remove power and power on.
- Check LED state.

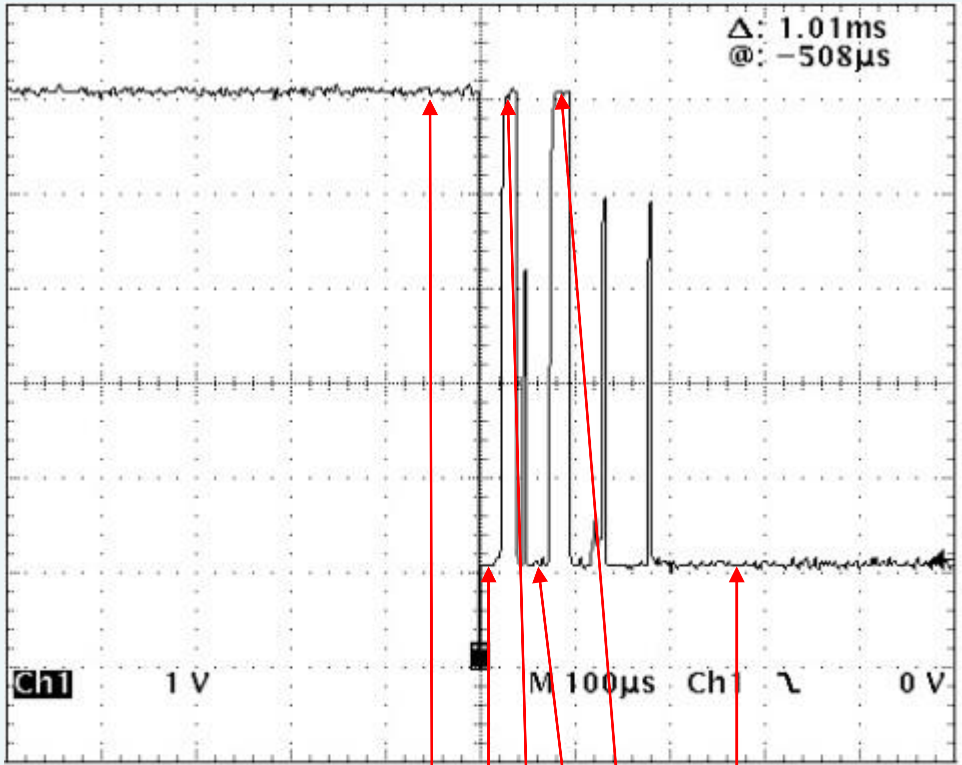
** Switch-Bouncing problem **



C.B.Choi gen1223@kau.ac.kr

Bouncing Problem for Switch

- **디지털 입력 – 스위치 - 디바운싱**



digitalRead(15) : H L H L H.. L

(Button 상태변화) & (변화후 50ms 이상 지속)

millis()… reset 후 1ms마다 1증가

예제 > Digital > Debounce

```
long lastDebounceTime = 0;
long debounceDelay = 50;
int lastButtonState = HIGH;
…
int reading = digitalRead(15);
if (reading != lastButtonState) {
   lastDebounceTime = millis();
}
if ((millis() - lastDebounceTime)
                > debounceDelay) {
   if (reading != buttonState) {
      buttonState = reading;
      Do Action…
   }
}
…
lastButtonState = reading;
…
```

Processor clock, Performance

- msec 당 실행명령 수

	아두이노 우노	라즈베리파이 모델 B
가격	30 달러	35 달러
크기	7.6 x 1.9 x 6.4 cm	8.6 x 5.4 x 1.7 cm
메모리	2 KB	512 MB
클럭 속도	16 MHz	700 MHz
네트워크	없음	10/100 유선 이더넷 RJ45
멀티태스킹	지원 안함	지원
입력 전압	7 ~ 12 V	5 V
Flash	32 KB	SD Card (2~16GB)
USB	1개, 입력 전용	2개
운영체제	없음	Linux 배포판 (기본은 Raspbian)
통합개발환경	Sketch, Scratch 사용 가능	Scratch, IDLE, Linux 지원 도구

Specifications - ESP32 DEVKIT V1 DOIT	
Number of cores	2 (Dual core)
Wi-Fi	2.4 GHz up to 150 Mbit/s
Bluetooth	BLE (Bluetooth Low Energy) and legacy Bluetooth
Architecture	32 bits
Clock frequency	Up to 240 MHz
RAM	512 KB
Pins	30
Peripherals	Capacitive touch, ADCs (analog-to-digital converter), DACs (digital-to-analog converter), I²C (Inter-Integrated Circuit), UART (universal asynchronous receiver/transmitter), CAN 2.0 (Controller Area Network), SPI (Serial Peripheral Interface), I²S (Integrated Inter-IC Sound), RMII (Reduced Media-Independent Interface), PWM (pulse width modulation), and more.

16MHz = (초당 16M 클럭) = (1만6천 클럭 / msec) = (1천600개 명령 / msec)

700MHz = (초당 700M 클럭) = (70만 클럭 / msec) = (7만개 명령 / msec)

240MHz = (초당 240M 클럭) = (24만 클럭 / msec) = (2만4천개 명령 / msec)

** 한 명령: 10개 클럭이라 가정함, MIPS = M instructions / sec

Remember Last GPIO State

<Task06-3>

```
#include <EEPROM.h>
// define the number of bytes you want to access
#define EEPROM_SIZE 1
const int buttonPin = 15;  // the number of the pushbutton pin
const int ledPin = 16;       // the number of the LED pin
// Variables will change:
int ledState = HIGH;      // the current state of the output pin
int buttonState;             // the current reading from the input pin
int lastButtonState = LOW; // the previous reading from the input pin
// the following variables are unsigned longs because the time, measured in
// milliseconds, will quickly become a bigger number than can be stored in an int.
unsigned long lastDebounceTime = 0; // the last time the output pin was toggled
unsigned long debounceDelay = 50;    // the debounce time; increase if the output flickers

void setup() {
            Serial.begin(115200);
// initialize EEPROM with predefined size
            EEPROM.begin(EEPROM_SIZE);
            pinMode(buttonPin, INPUT);
            pinMode(ledPin, OUTPUT);
// read the last LED state from flash memory
            ledState = EEPROM.read(0);
// set the LED to the last stored state
            digitalWrite(ledPin, ledState);
}
```

Remember Last GPIO State

<Task06-3>

```
void loop() {
// read the state of the switch into a local variable:
        int reading = digitalRead(buttonPin);
// check to see if you just pressed the button
// (i.e. the input went from LOW to HIGH), and you've waited long enough
// since the last press to ignore any noise:
// If the switch changed, due to noise or pressing:
        if (reading != lastButtonState) {
        // reset the debouncing timer
                lastDebounceTime = millis();
        }
        if ((millis() - lastDebounceTime) > debounceDelay) {
        // whatever the reading is at, it's been there for longer than the debounce
        // delay, so take it as the actual current state:
        // if the button state has changed:
                if (reading != buttonState) {
                        buttonState = reading;
                        // only toggle the LED if the new button state is HIGH
                        if (buttonState == HIGH) {
                                ledState = !ledState;
                        }
                }
        }
```

Remember Last GPIO State

<Task06-3>

```
// save the reading. Next time through the loop, it'll be the lastButtonState:
        lastButtonState = reading;
// if the ledState variable is different from the current LED state
        if (digitalRead(ledPin)!= ledState) {
                Serial.println("State changed");
                // change the LED state
                digitalWrite(ledPin, ledState);
                // save the LED state in flash memory
                EEPROM.write(0, ledState);
                EEPROM.commit();
                Serial.println("State saved in flash memory");
        }
}
```