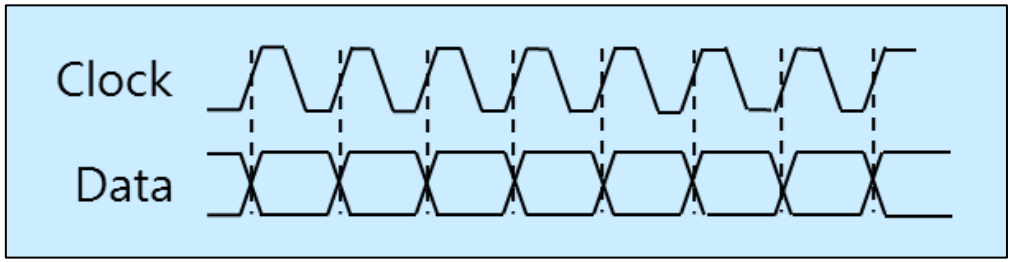
바이브코딩을 이용한 사물인터넷

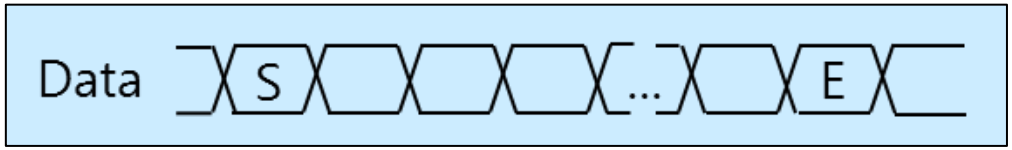
- Serial Communication: I2C and SPI

Serial Communication

- Parallel communication – transmits multiple bits simultaneously; needs multiple data bit lines
- Serial communication – transmits one bit at one time
  - slow but needs small number of lines for transmission
  - several interface for serial communication: I2C, SPI, UART
  - synchronous: uses clock lines for synchronization
    - I2C, SPI



  - asynchronous: no clock lines for synchronization
    - UART



I2C Communication

- I2C communication
- I2C in Raspberry Pi

I2C Communication

- Inter-Integrated Circuit (I2C)
  - master-slave, serial communication bus developed by Philips Semiconductor(NXP)
  - Typically used for attaching lower-speed peripheral ICs to processors and microcontrollers in short-distance
  - I²C uses only two bidirectional lines, Serial Data Line (SDA) and Serial Clock Line (SCL)
  - I2C has a 7-bit or a 10-bit (depending on the device used) address (ID) space
  - Typical speed: 10kbps(low speed mode), 100 kbps (standard mode), 400 kbps (fast mode)

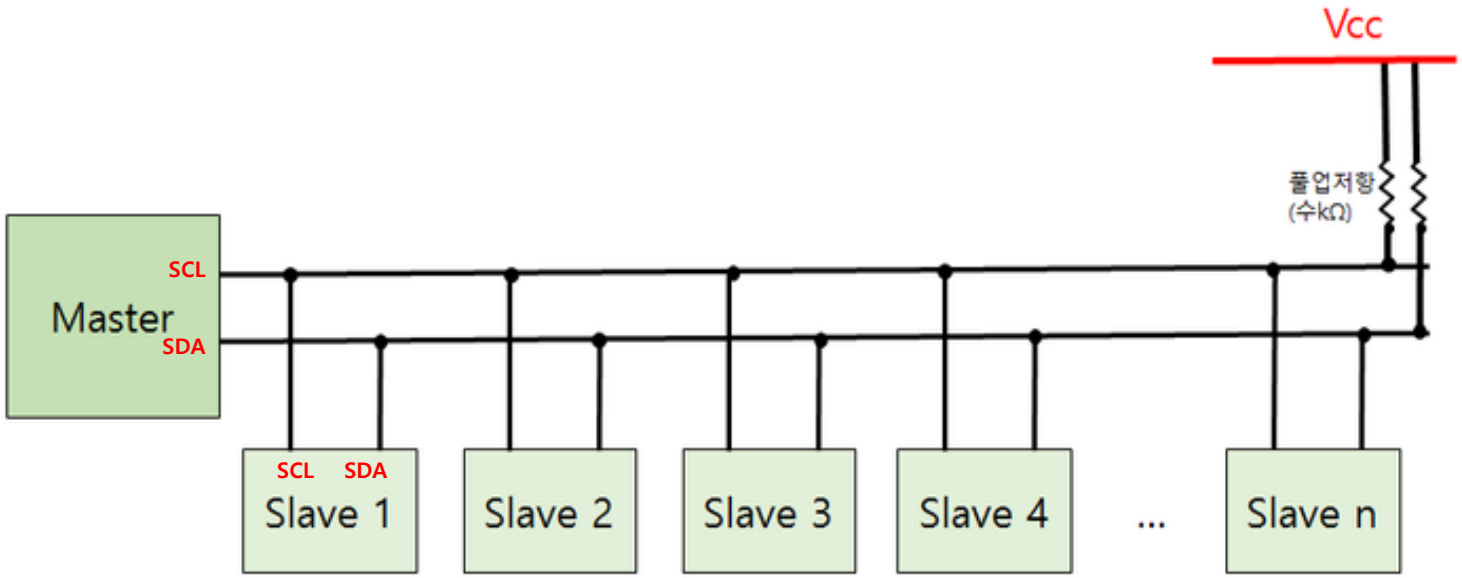
Raspberry Pi I2C

- Raspberry Pi I2C pins

WPI	BCM/Name	물리적 구성(핀)		BCM/Name	WPI
	3.3V	1	2	5V	
8	GPIO 2/I2C SDA	3	4	5V	
9	GPIO 3/I2C SCL	5	6	Ground	
7	GPIO 4	7	8	GPIO 14/TXD	15
	Ground	9	10	GPIO 15/RXD	16
0	GPIO 17	11	12	GPIO 18	1
2	GPIO 27	13	14	Ground	
3	GPIO 22	15	16	GPIO 23	4
	3.3 V	17	18	GPIO 24	5
12	GPIO 10/SPI MOSI	19	20	Ground	
13	GPIO 9/SPI MISO	21	22	GPIO 25	6
14	GPIO 11/SPI CLK	23	24	GPIO 8/SPI_CE0	10
	Ground	25	26	GPIO 7/SPI_CE1	11
30	ID_SD	27	28	ID_SC	31
21	GPIO 5	29	30	Ground	
22	GPIO 6	31	32	GPIO 12	26
23	GPIO 13	33	34	Ground	
24	GPIO 19	35	36	GPIO 16	27
25	GPIO 26	37	38	GPIO 20	28
	Ground	39	40	GPIO 21	29

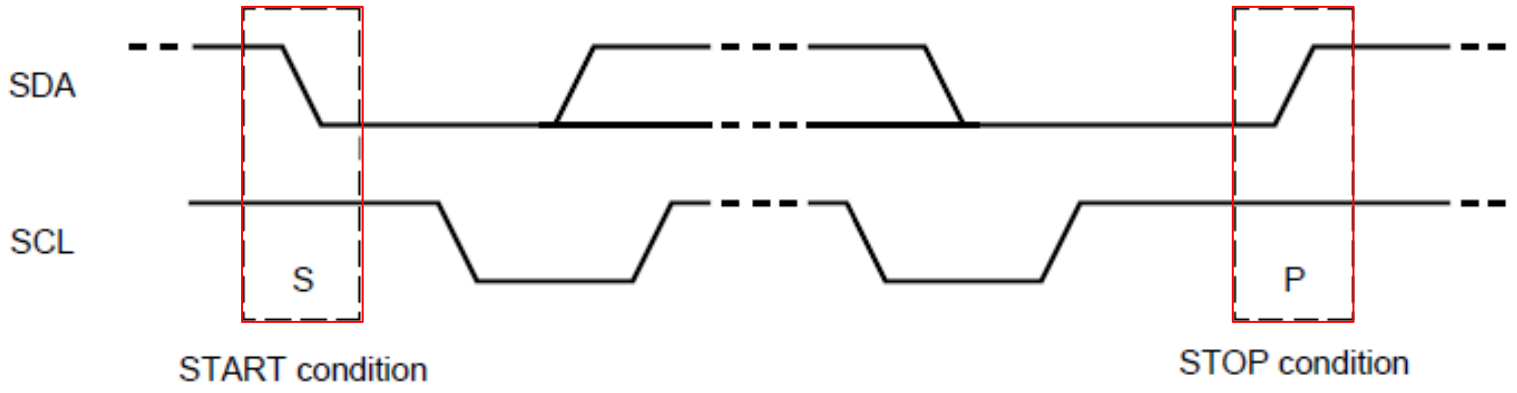
I2C Introduction

- I2C Bus Applications



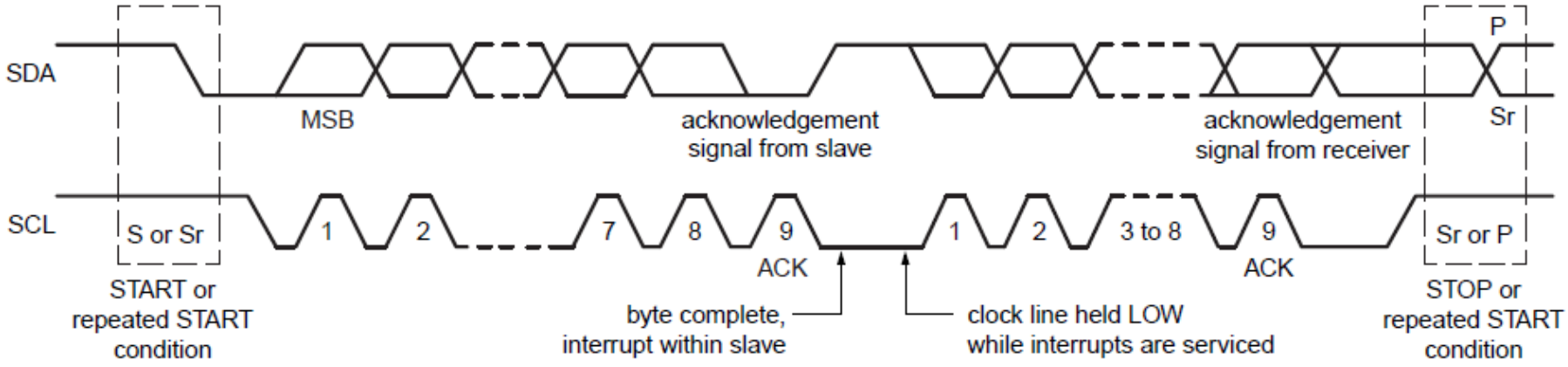
I2C Introduction

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



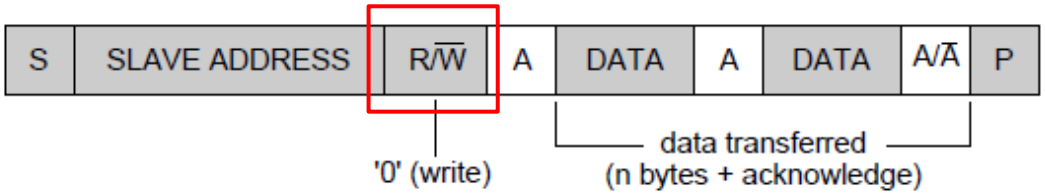
I2C Introduction

- 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 byte of data, it can hold SCL LOW to force the master into a wait state



I2C Introduction

- I2C Communication: Byte Format
  - A master addresses a slave with a 7-bit address

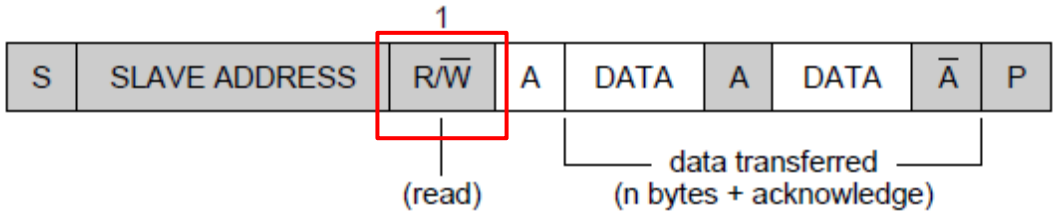


from master to slave

from slave to master

A = acknowledge (SDA LOW)
Ā = not acknowledge (SDA HIGH)
S = START condition
P = STOP condition

  - A master reads DATA from a slave after the first byte



I2C Communication

- Configuration for I2C related software
  - enable I2C using sudo raspi-config and reboot
  - check installed kernel modules : lsmod | grep i2c

```
$ lsmod | grep i2c
i2c-dev
i2c-bcm2835
```

I2C Communication

- Configuration for I2C related software
  - check installed packages:

```
$ apt list --installed | grep i2c-tools
$ apt list --installed | grep smbus
```

  - if not installed, install i2c-tools and python3-smbus packages

```
pi@raspberrypi ~ $ sudo apt install i2c-tools
pi@raspberrypi ~ $ sudo apt install python3-smbus
```

I2C Communication

- i2ctools
  - i2cdetect : used to detect i2c slave devices on i2c-bus
  - i2cdump : used to get all register values of i2c device (i2c-address) attached on i2c-bus
  - i2cget : used to get a register value of i2c device (i2c-address) attached on i2c-bus
  - i2cset : used to set a value to a register of i2c device (i2c-address) attached on i2c-bus
  - i2ctransfer : used to get a value after setting a value to a register of i2c device attached on i2c-bus

I2C Communication

- python3-smbus module
  - python module for controlling i2c device (/dev/i2c-1) in RaspberryPi
  - install: sudo apt install python3-smbus

  - import smbus
  - dev = smbus.SMBus(1) # open /dev/i2c-1

I2C Communication

- python3-smbus module
  - dev.read_byte_data(dev_addr, reg_addr) # read 1B
  - dev.write_byte_data(dev_addr, reg_addr, val) # write 1B
  - dev.read_word_data(dev_addr, reg_addr) # read 2B
  - dev.write_word_data(dev_addr, reg_addr, val) # write 2B
  - dev.read_i2c_block_data(dev_addr, reg_addr, length)
  - dev.write_i2c_block_data(dev_addr, reg_addr, val_list)

mlx90614 센서

- mlx90614 온도센서:
  - 비접촉식 적외선 온도센서: Melexis 사
  - 인터페이스: i2c, pwm
  - 측정범위 : -70~+380 °C
  - i2c address: 0x5a
  - 물체 온도값 제공: 16-bit

  ambient temperature(address 0x06),
  object temperature (address 0x07)

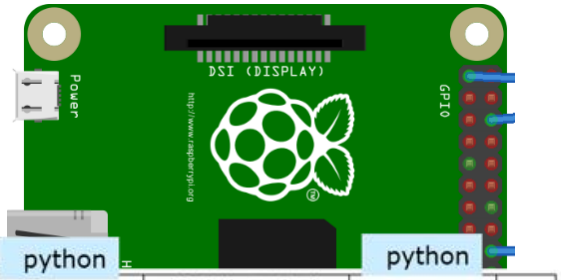
mlx90614 센서

- mlx90614 온도센서:
  - RAM address:

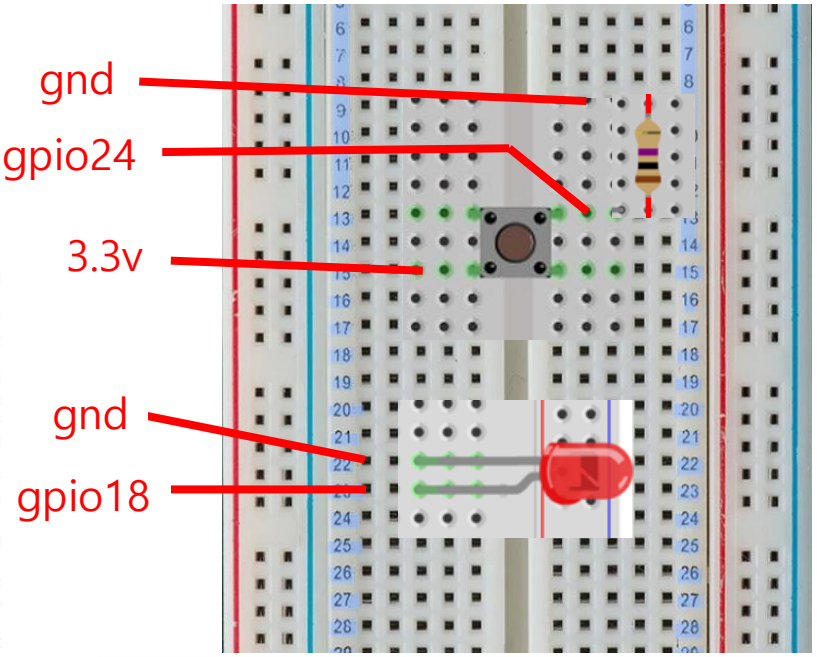
RAM (32x16)		
Name	Address	Read access
Melexis reserved	0x00	Yes
…	…	…
Melexis reserved	0x03	Yes
Raw data IR channel 1	0x04	
Raw data IR channel 2	0x05	
T_A	0x06	Yes
T_{OBJ1}	0x07	Yes
T_{OBJ2}	0x08	Yes
Melexis reserved	0x09	Yes
…	…	…
Melexis reserved	0x1F	Yes

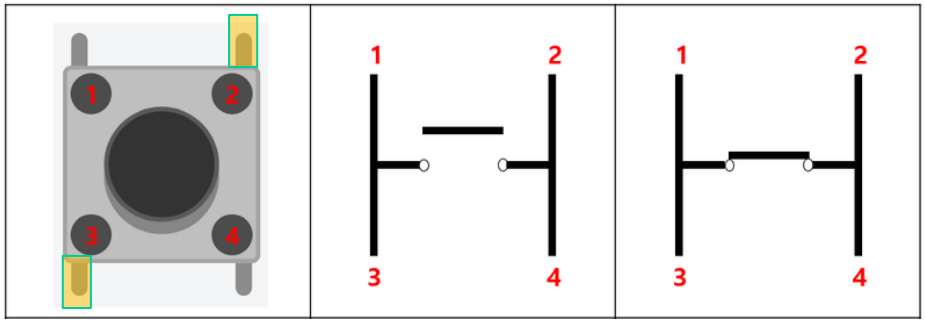
사물인터넷 프로그래밍: I2C 센서

- I2C 프로그래밍: 버튼, led 연결



WPI	BCM/Name	물리적 구성(핀)		BCM/Name	WPI
	3.3V	1	2	5V	
8	GPIO 2/I2C SDA	3	4	5V	
9	GPIO 3/I2C SCL	5	6	Ground	
7	GPIO 4	7	8	GPIO 14/TXD	15
	Ground	9	10	GPIO 15/RXD	16
0	GPIO 17	11	12	GPIO 18	1
2	GPIO 27	13	14	Ground	
3	GPIO 22	15	16	GPIO 23	4
	3.3 V	17	18	GPIO 24	5
12	GPIO 10/SPI MOSI	19	20	Ground	
13	GPIO 9/SPI MISO	21	22	GPIO 25	6
14	GPIO 11/SPI CLK	23	24	GPIO 8/SPI_CE0	10
	Ground	25	26	GPIO 7/SPI_CE1	11
30	ID_SD	27	28	ID_SC	31
21	GPIO 5	29	30	Ground	
22	GPIO 6	31	32	GPIO 12	26
23	GPIO 13	33	34	Ground	
24	GPIO 19	35	36	GPIO 16	27
25	GPIO 26	37	38	GPIO 20	28
	Ground	39	40	GPIO 21	29





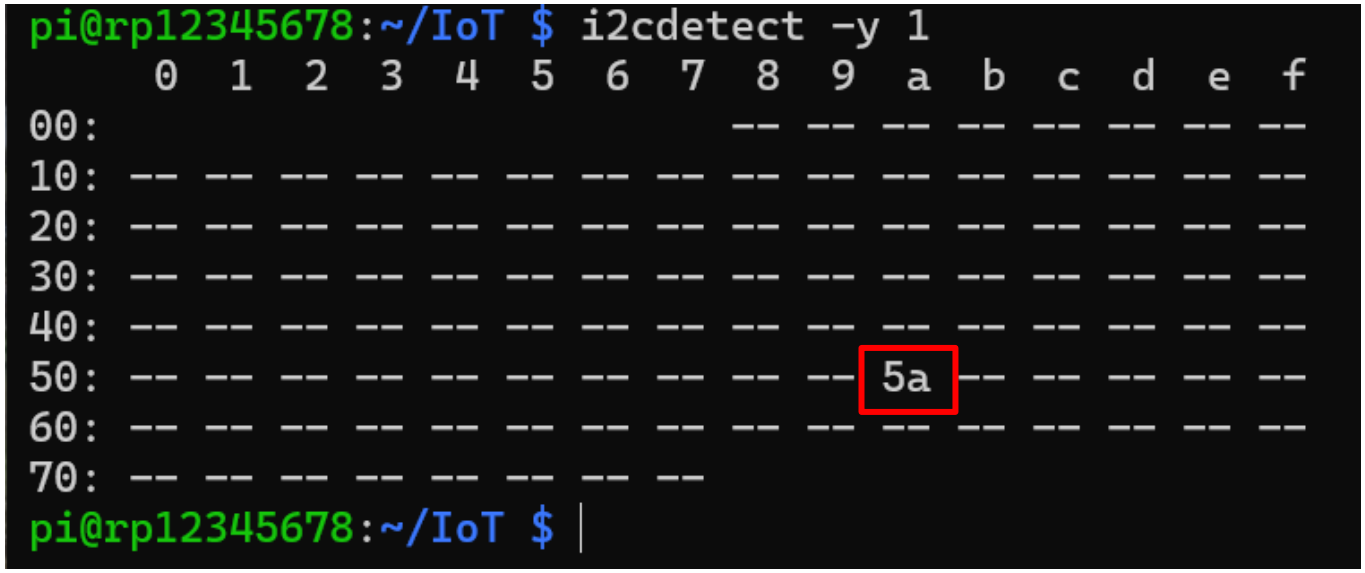
사물인터넷 프로그래밍: I2C 센서

- I2C 센서 프로그래밍 : mlx90614 연결

WPI	BCM/Name	물리적 구성(핀)		BCM/Name	WPI
	3.3V	1	2	5V	
8	GPIO 2/I2C SDA	3	4	5V	
9	GPIO 3/I2C SCL	5	6	Ground	
7	GPIO 4	7	8	GPIO 14/TXD	15
	Ground	9	10	GPIO 15/RXD	16
0	GPIO 17	11	12	GPIO 18	1
2	GPIO 27	13	14	Ground	
3	GPIO 22	15	16	GPIO 23	4
	3.3 V	17	18	GPIO 24	5
12	GPIO 10/SPI MOSI	19	20	Ground	
13	GPIO 9/SPI MISO	21	22	GPIO 25	6
14	GPIO 11/SPI CLK	23	24	GPIO 8/SPI_CE0	10
	Ground	25	26	GPIO 7/SPI_CE1	11
30	ID_SD	27	28	ID_SC	31
21	GPIO 5	29	30	Ground	
22	GPIO 6	31	32	GPIO 12	26
23	GPIO 13	33	34	Ground	
24	GPIO 19	35	36	GPIO 16	27
25	GPIO 26	37	38	GPIO 20	28
	Ground	39	40	GPIO 21	29

I2C Communication

- RaspberryPi i2ctools
  - i2cdetect : i2cdetect –y 1



사물인터넷 프로그래밍: I2C 센서

- I2C 센서 프로그래밍 :

현재 rp84041 이라는 라즈베리파이 호스트가 연결되어 있고, 이 라즈베리파이에는 아래와 같은 센서들이 연결되어 있어.
[led] gpio18 번에 연결,
[버튼] 풀다운 저항을 통해 gpio24 번에 연결,
[mlx90614 센서] mlx90614 온도센서가 i2c를 통해 연결됨,
이 상태에서 아래와 같이 동작하는 프로그램을 만들어줘.
[동작 모드] 중지모드, 감지모드 (초기상태는 중지모드), 인터럽트를 통해 버튼 입력을 감지하고, 버튼을 눌러지면 동작모드는 바뀜,
중지모드에서는 모든 동작이 멈춰있고, 감지모드에서는 3초 간격으로 온도센서로부터 현재온도를 읽고 값을 출력, 현재온도값이 25도C 이상이면 led를 on 상태로 함.

연결된 rp84041 호스트의 pi 게정의 홈폴더에 IoT 라는 폴더가 있어. 프로그램을 이 IoT 폴더에 복사해줘.

사물인터넷 프로그래밍: I2C 센서

- I2C 센서 프로그래밍 : 실행

```
pi@rp84041:~/IoT $ python3 sensor_monitor.py
[시작] 초기 동작 모드: 중지모드
[버튼] 동작 모드 변경: 감지모드
[감지] 현재온도: 26.03 °C, LED: ON
[버튼] 동작 모드 변경: 중지모드
[버튼] 동작 모드 변경: 감지모드
[감지] 현재온도: 26.37 °C, LED: ON
[감지] 현재온도: 25.11 °C, LED: ON
[감지] 현재온도: 21.61 °C, LED: OFF
[감지] 현재온도: 20.99 °C, LED: OFF
[감지] 현재온도: 25.59 °C, LED: ON
[감지] 현재온도: 22.81 °C, LED: OFF
[감지] 현재온도: 25.73 °C, LED: ON
[감지] 현재온도: 26.31 °C, LED: ON
[버튼] 동작 모드 변경: 중지모드
[버튼] 동작 모드 변경: 감지모드
[감지] 현재온도: 26.21 °C, LED: ON
[감지] 현재온도: 26.35 °C, LED: ON
[버튼] 동작 모드 변경: 중지모드
```

사물인터넷 프로그래밍: I2C 센서

- I2C 센서 구현 프로그램

```python
#!/usr/bin/env python3
"""Button-controlled MLX90614 temperature monitor for Raspberry Pi."""

from __future__ import annotations

import signal
import sys
import threading
import time
from enum import Enum

try:
    import RPi.GPIO as GPIO
    from smbus2 import SMBus
except ImportError as exc:
    missing = exc.name or "required Raspberry Pi package"
    print(
        f"Missing dependency: {missing}\n"
        "Install dependencies on the Raspberry Pi with:\n"
        "  python3 -m pip install -r requirements.txt",
        file=sys.stderr,
    )
    raise SystemExit(1) from exc
```

사물인터넷 프로그래밍: I2C 센서

- I2C 센서 구현 프로그램

```
LED_PIN = 18
BUTTON_PIN = 24

I2C_BUS = 1
MLX90614_ADDRESS = 0x5A
MLX90614_OBJECT_TEMP_REGISTER = 0x07

TEMP_INTERVAL_SECONDS = 3.0
LED_THRESHOLD_C = 25.0
BUTTON_BOUNCE_MS = 250


class Mode(Enum):
    STOPPED = "중지모드"
    DETECTING = "감지모드"


mode = Mode.STOPPED
mode_lock = threading.Lock()
mode_changed = threading.Event()
running = threading.Event()
running.set()
```

사물인터넷 프로그래밍: I2C 센서

I2C 센서 프로그래밍 : 구현 프로그램

```
def read_mlx90614_temperature_c(bus: SMBus) -> float:
    """Read object temperature from MLX90614 and convert Kelvin units to Celsius."""
    raw = bus.read_word_data(MLX90614_ADDRESS, MLX90614_OBJECT_TEMP_REGISTER)
    return raw * 0.02 - 273.15


def setup_gpio() -> None:
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED_PIN, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.add_event_detect(
        BUTTON_PIN,
        GPIO.RISING,
        callback=handle_button_press,
        bouncetime=BUTTON_BOUNCE_MS,
    )
```

사물인터넷 프로그래밍: I2C 센서

- I2C 센서 프로그래밍 : 구현 프로그램

```
def handle_button_press(_: int) -> None:
    global mode

    with mode_lock:
        mode = Mode.DETECTING if mode is Mode.STOPPED else Mode.STOPPED
        current_mode = mode

    if current_mode is Mode.STOPPED:
        GPIO.output(LED_PIN, GPIO.LOW)

    print(f"[버튼] 동작 모드 변경: {current_mode.value}", flush=True)
    mode_changed.set()
```

사물인터넷 프로그래밍: I2C 센서

- I2C 센서 프로그래밍 : 구현 프로그램

```
def get_mode() -> Mode:
    with mode_lock:
        return mode


def request_exit(signum: int, _: object) -> None:
    print(f"\n[종료] signal {signum} 수신", flush=True)
    running.clear()
    mode_changed.set()


def wait_for_next_cycle(seconds: float) -> None:
    mode_changed.wait(seconds)
    mode_changed.clear()
```

사물인터넷 프로그래밍: I2C 센서

I2C 센서 프로그래밍 : 구현 프로그램

```
def main() -> int:
    signal.signal(signal.SIGINT, request_exit)
    signal.signal(signal.SIGTERM, request_exit)

    setup_gpio()
    print("[시작] 초기 동작 모드: 중지모드", flush=True)

    try 문: main loop ; next page

    finally:
        GPIO.output(LED_PIN, GPIO.LOW)
        GPIO.cleanup()
        print("[정리] GPIO 정리 완료", flush=True)

    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

사물인터넷 프로그래밍: I2C 센서

```
try:
    with SMBus(I2C_BUS) as bus:
        while running.is_set():
            if get_mode() is Mode.STOPPED:
                GPIO.output(LED_PIN, GPIO.LOW)
                wait_for_next_cycle(TEMP_INTERVAL_SECONDS)
                continue

            try:
                temperature_c = read_mlx90614_temperature_c(bus)
            except OSError as exc:
                print(f"[오류] MLX90614 읽기 실패: {exc}", file=sys.stderr, flush=True)
                GPIO.output(LED_PIN, GPIO.LOW)
            else:
                led_on = temperature_c >= LED_THRESHOLD_C
                GPIO.output(LED_PIN, GPIO.HIGH if led_on else GPIO.LOW)
                led_state = "ON" if led_on else "OFF"
                print(
                    f"[감지] 현재온도: {temperature_c:.2f} °C, LED: {led_state}",
                    flush=True,
                )

            wait_for_next_cycle(TEMP_INTERVAL_SECONDS)
```