임베디드 시스템 기말 프로젝트

IOT 자판기

목차

문제 상황

자판기는 일상생활에서 쉽게 볼 수 있는 편의 장치 중 하나이다. 대부분의 자판기를 보면 재고가 남아 있지만, 사람들이 많이 다니지 않는 곳의 자판기를 보면 재고가 없을 때가 많다. 이는 수요가 없는 지역에서 자판기의 재고 관리에 쓰는 노력이 수익대비 크기 때문이다. 그래서 사람이 일일이 확인하러 다니지 않고도 온라인 database 상으로 재고를 관리할 수 있는 IOT를 접목한 자판기를 고안하게 되었다.

또한, 외진 곳은 CCTV나 경비가 부족하다. 이러한 이유로 자판기의 물건이 도난당하는 일이 있는 데, 싸이렌을 울리며 온라인 database상으로 관리자에게 상황을 알리는 기능을 추가하여 이에 대비할 수 있다.

기대 효과

1) 자판기를 IOT로 구현함으로써, 재고가 다 떨어진 상황을 외부에서 확인할 수 있다. 즉, 계속해서 재고를 확인하는 인력을 줄일 수 있다.

2) 현재까지의 자판기 수입금을 확인하여 이후 자판기 투자(제품 선택, 자판기 위치 선택 등)에 대한 계획을 세울 수 있다.

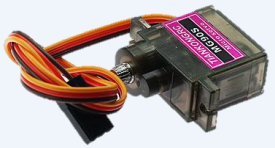
3) 자판기의 물건을 훔치는 일을 막아, CCTV나 경비가 부족한 지역에서도 사용할 수 있다는 장점이 있다.

4) 자판기에 기본적으로 탑재되어 있는 관리자 호출 버튼을 이용하여 자판기에 문제가 발생하였을 때 대체가 가능하다.

HW 구성

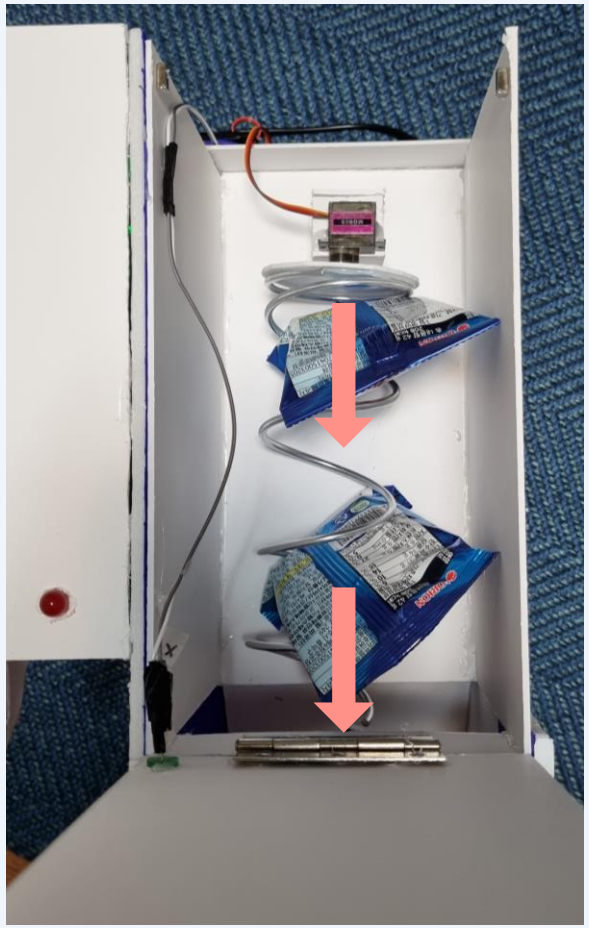
HW 구성 - 윗면

상품 출력용
360도 서보 모터



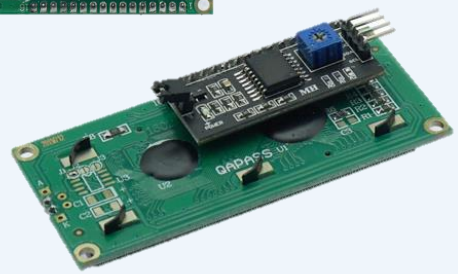
HW 구성 - 윗면

상품 출력의 원리



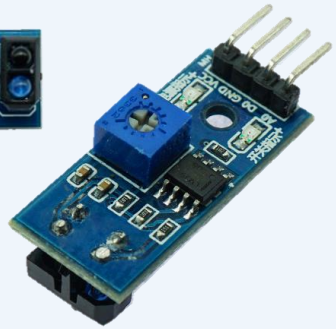
HW 구성 - 전면

system 출력부



LCD1602 I2C 모듈

구매/반환/호출 버튼



TCRT5000
적외선 감지 센서 모듈

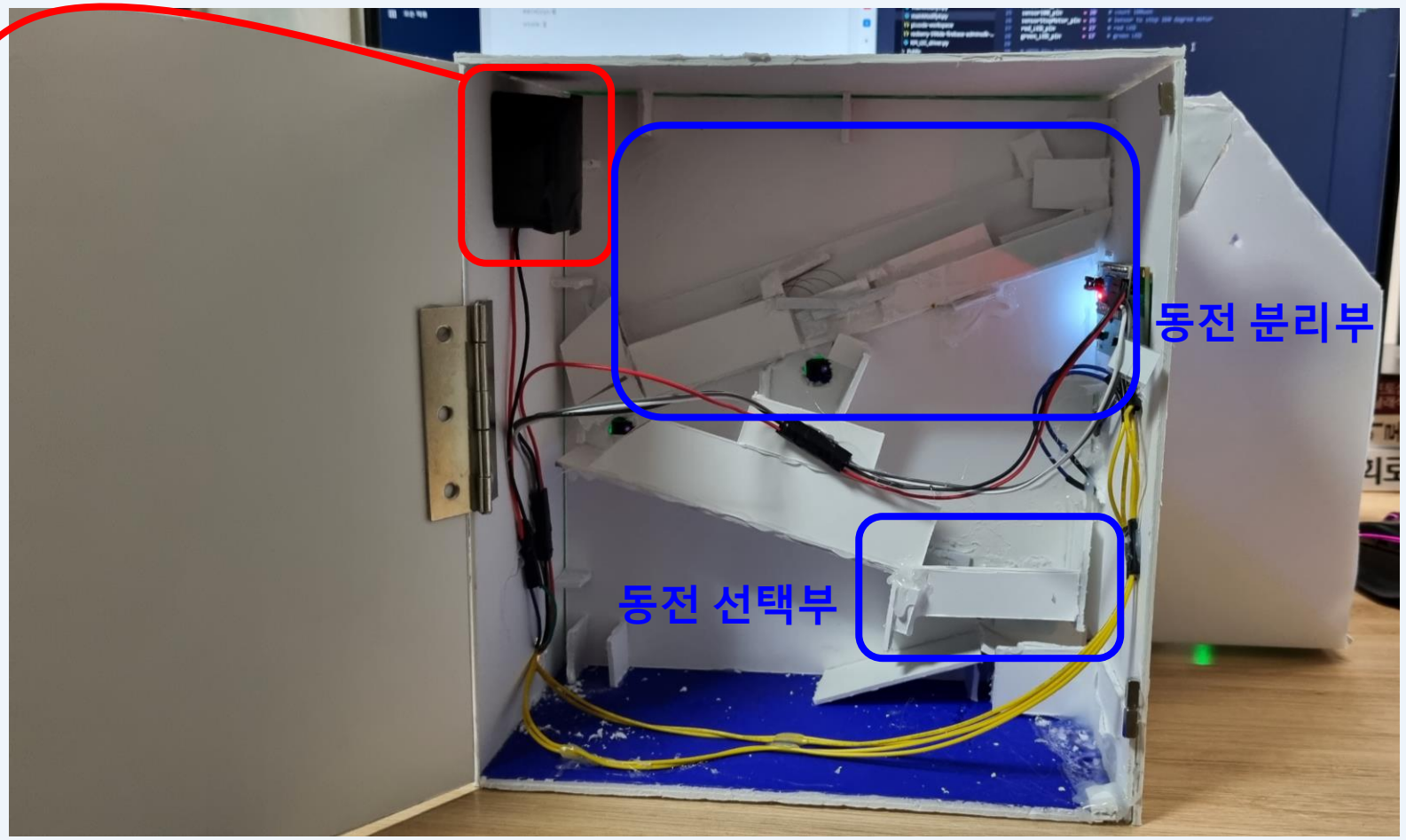
HW 구성 - 전면

상품 출력 LED & 경고 LED

HW 구성 - 좌측면

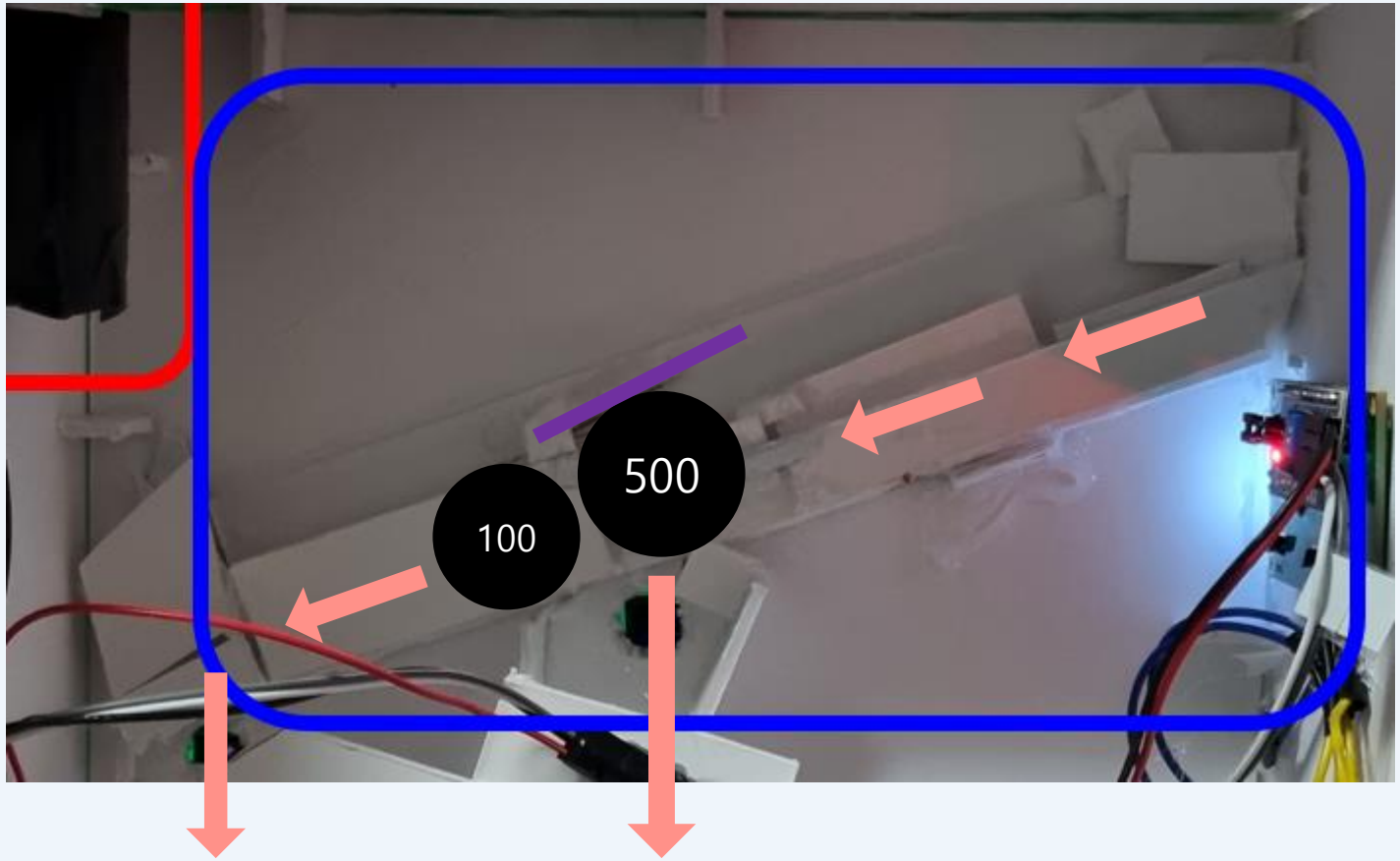
system 출력부

HW 구성 - 좌측면



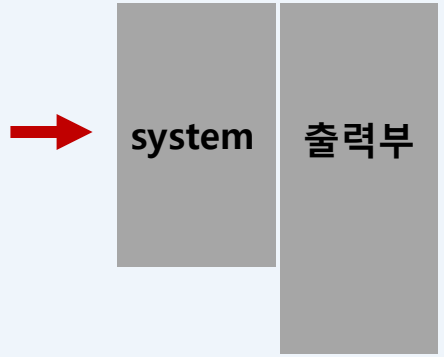
HW 구성 - 좌측면

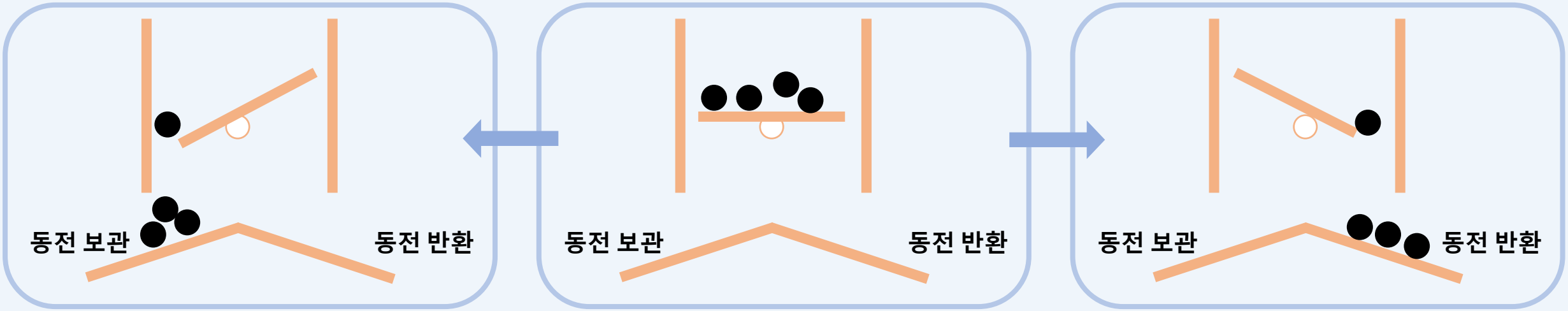
동전 분류의 원리



HW 구성 - 좌측면

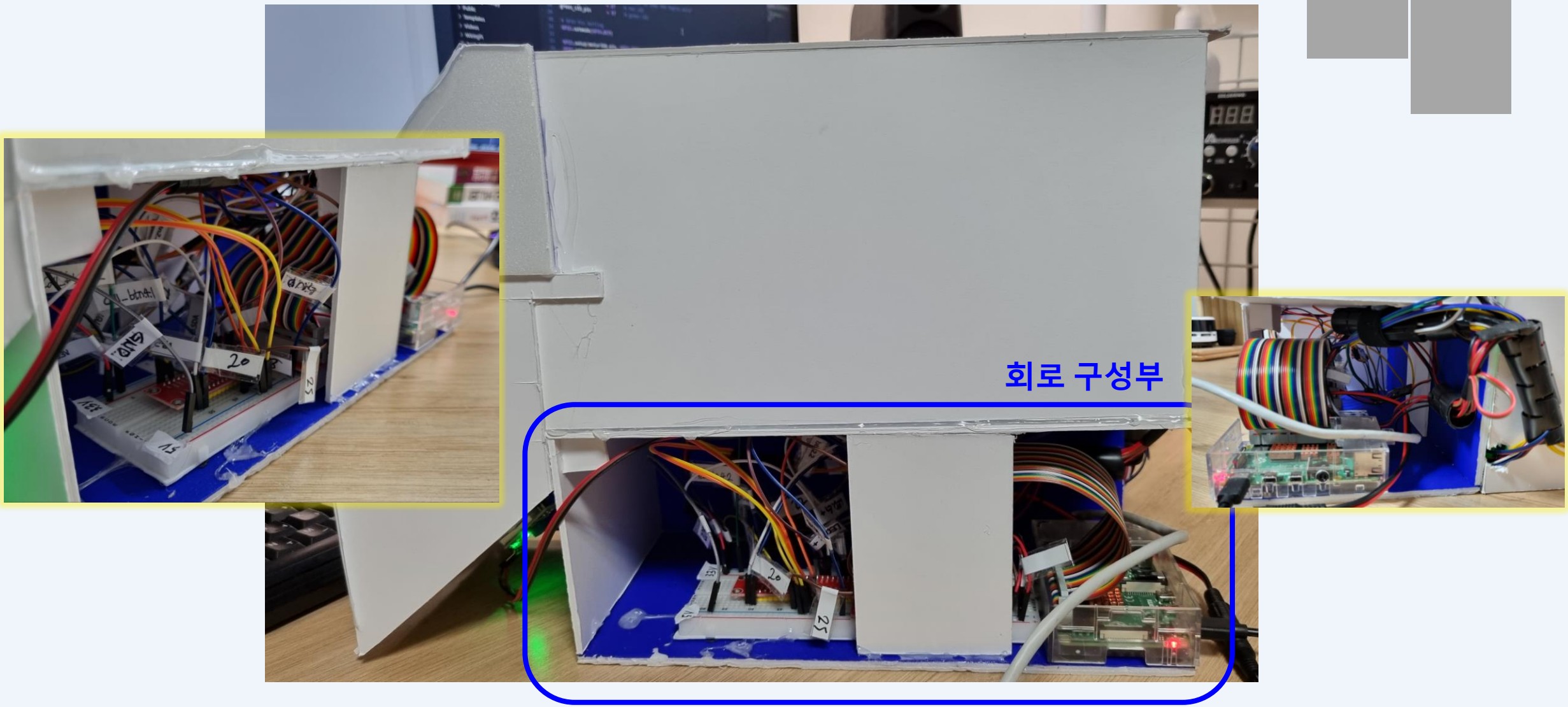
동전 반환의 원리



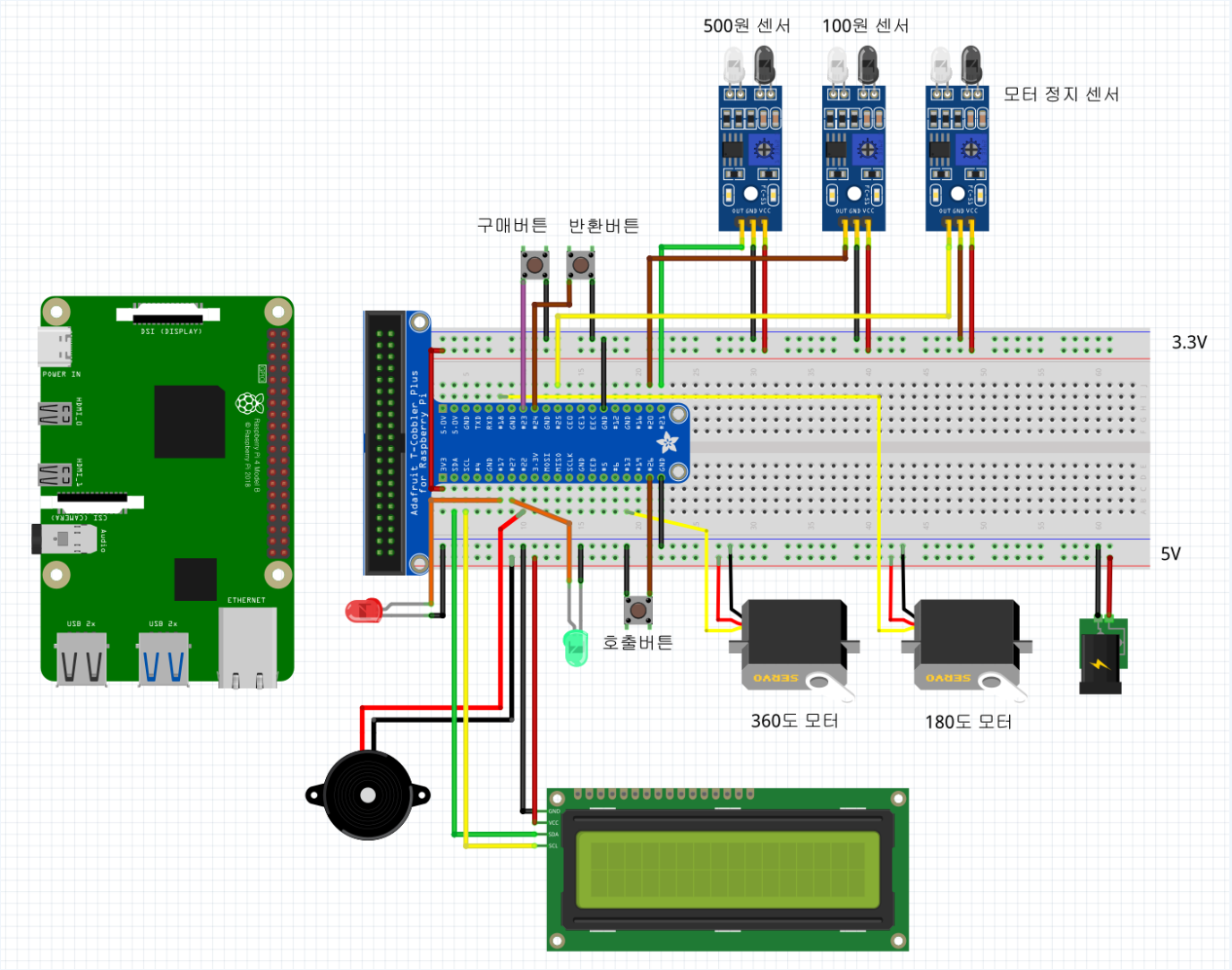


동전 보관 대기 상태 대기 상태

HW 구성 - 우측면

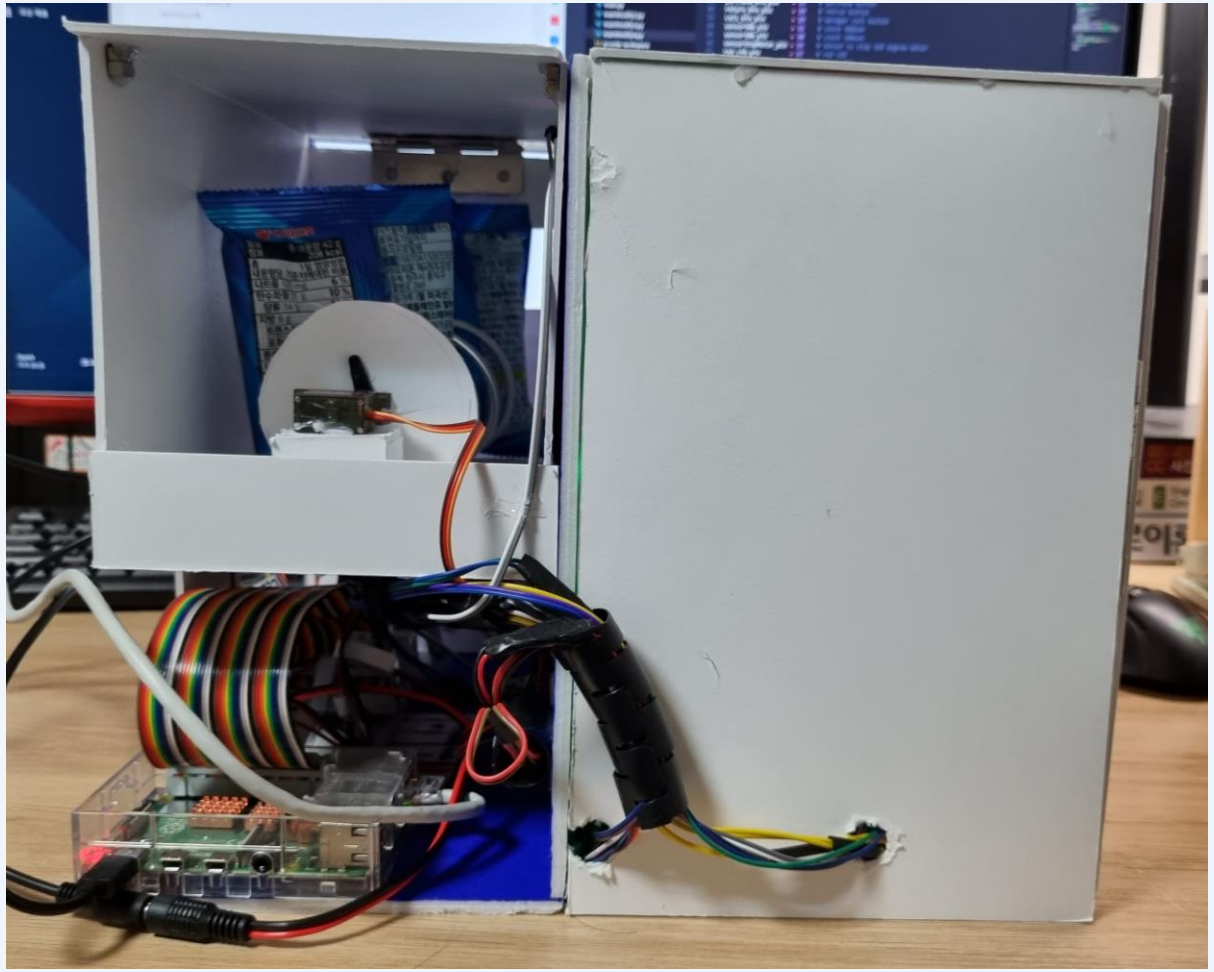


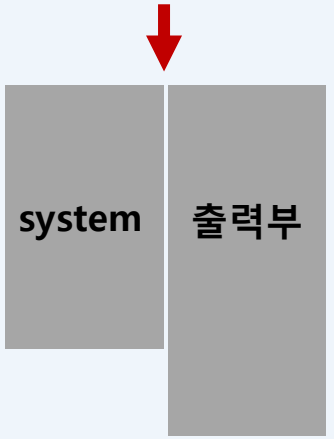
HW 구성 - 우측면



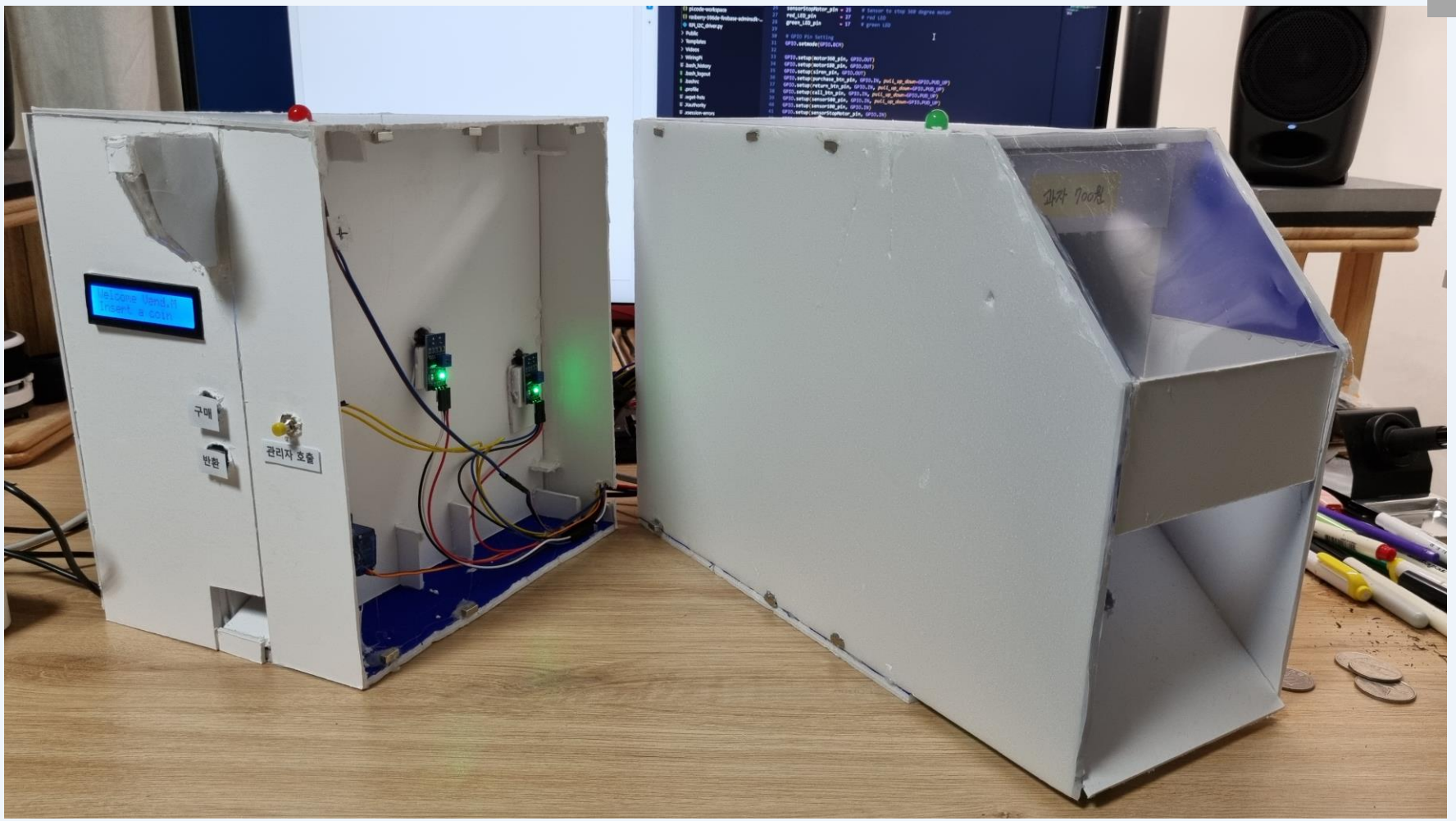
핀 구성

HW 구성 - 후면

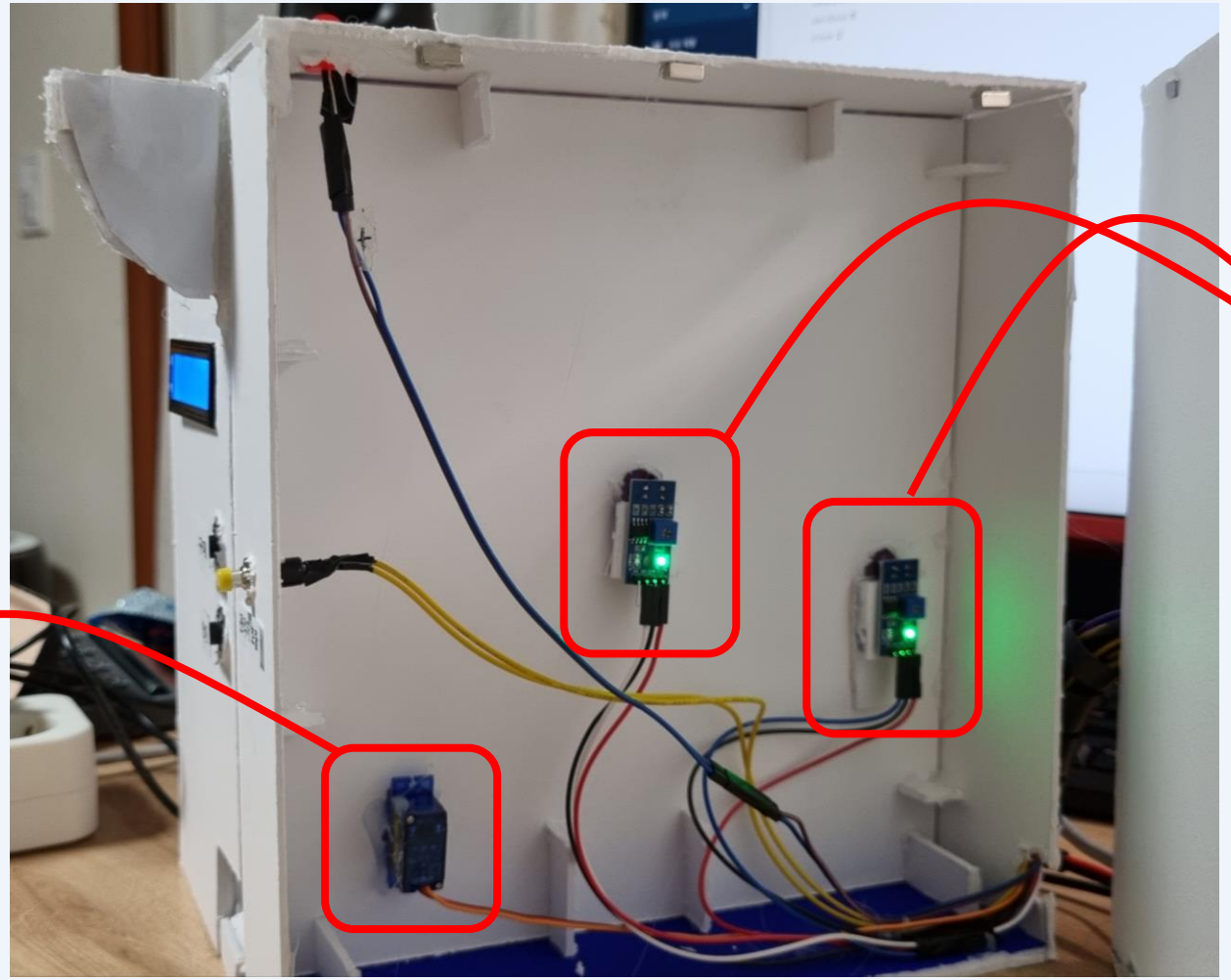




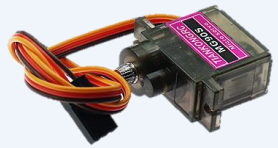
HW 구성 - 중간면

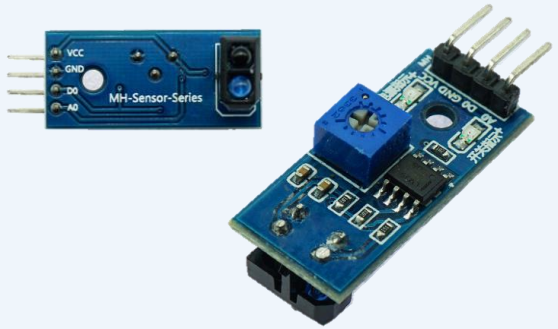


HW 구성 - 중간면



동전 분류용
180도 서보 모터





TCRT5000
적외선 감지 센서 모듈

SW 구성

SW 설계

- 설계언어: 파이썬
- 외부라이브러리: python time 모듈, firebase 모듈, GPIO 모듈, LCD 모듈
- 외부 센서나 버튼으로부터의 신호는 인터럽트로 처리, 메인 함수는 Polling으로 구현
- firebase 연동을 통해 온라인 상으로 자판기 관리가 가능

SW 구성

외부라이브러리

외부 라이브러리 import

Sleep() 함수를 쓰기 위해 import

```
import time
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db

import RPi.GPIO as GPIO
import RPi_I2C_driver as I2C
```

Firebase에 데이터를 읽고 쓰기 위해서 import

라즈베리 파이 GPIO를 사용하기 위한 모듈

I2C를 사용하여 LCD를 사용하기 위한 모듈

외부 라이브러리 - Firebase Setting

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
```

- Python
  - Overview
  - firebase_admin
  - firebase_admin.app_check
  - firebase_admin.auth
  - firebase_admin.credentials
  - firebase_admin.db
  - firebase_admin.exceptions
  - firebase_admin.storage
  - firebase_admin.firestore
  - firebase_admin.firestore_async
  - firebase_admin.instance_id
  - firebase_admin.messaging
  - firebase_admin.project_management
  - firebase_admin.ml
  - firebase_admin.tenant_mgt

출처: Google Firebase Documentation

외부 라이브러리 - Firebase Setting

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
```

get_credential()

Returns the Google credential instance used for authentication.

class firebase_admin.credentials.Certificate(*cert*)

Bases: Base

A credential initialized from a JSON certificate keyfile.

get_credential()

Returns the underlying Google credential.

Returns:	A Google Auth credential instance.
Return type:	google.auth.credentials.Credentials

property project_id

property service_account_email

property signer

JSON(제이슨[1], JavaScript Object Notation)은 속성-값 쌍(attribute–value pairs), 배열 자료형(array data types) 또는 기타 모든 시리얼화 가능한 값(serializable value) 또는 "키-값 쌍"으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷이다. 비동기 브라우저/서버 통신 (AJAX)을 위해, 넓게는 XML(AJAX가

```
######### Firebase Setting ##########
cred = credentials.Certificate("/home/pi/Project/rasberry-596de-firebase-adminsdk-sm1ch-a946597156.json")
firebase_admin.initialize_app(cred,{
    'databaseURL' : 'https://rasberry-596de-default-rtdb.firebaseio.com/'
})
ref = db.reference() #기본 위치 지정
#####################################
```

출처: Google Firebase Documentation, 위키 백과 JSON

외부 라이브러리 - Firebase Setting

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
```

firebase_admin.initialize_app(*credential=None, options=None, name='[DEFAULT]'*)

Initializes and returns a new App instance.

Creates a new App instance using the specified options and the app name. If an instance already exists by the same app name a ValueError is raised. If options are not provided an attempt is made to load the options from the environment. This is done by looking up the `FIREBASE_CONFIG` environment variable. If the value of the variable starts with "{", it is parsed as a JSON object. Otherwise it is treated as a file name and the JSON content is read from the corresponding file. Use this function whenever a new App instance is required. Do not directly invoke the App constructor.

Parameters:	• **credential** – A credential object used to initialize the SDK (optional). If none is provided, Google Application Default Credentials are used. • **options** – A dictionary of configuration options (optional). Supported options include `databaseURL`, `storageBucket`, `projectId`, `databaseAuthVariableOverride`, `serviceAccountId` and `httpTimeout`. If `httpTimeout` is not set, the SDK uses a default timeout of 120 seconds. • **name** – Name of the app (optional).
Returns:	A newly initialized instance of App.
Return type:	App
Raises:	**ValueError** – If the app name is already in use, or any of the provided arguments are invalid.

```
########## Firebase Setting ##########
cred = credentials.Certificate("/home/pi/Project/rasberry-596de-firebase-adminsdk-sm1ch-a946597156.json")
firebase_admin.initialize_app(cred,{
    'databaseURL' : 'https://rasberry-596de-default-rtdb.firebaseio.com/'
})
ref = db.reference() #기본 위치 지정
######################################
```

출처: Google Firebase Documentation

외부 라이브러리 - Firebase Setting

```
import firebase_admin
from firebase_admin import credentials
from firebase_admin import db
```

`firebase_admin.db.reference(path='/', app=None, url=None)`

Returns a database `Reference` representing the node at the specified path.

If no path is specified, this function returns a `Reference` that represents the database root. By default, the returned References provide access to the Firebase Database specified at app initialization. To connect to a different database instance in the same Firebase project, specify the `url` parameter.

Parameters:	• **path** – Path to a node in the Firebase realtime database (optional). • **app** – An App instance (optional). • **url** – Base URL of the Firebase Database instance (optional). When specified, takes precedence over the the `databaseURL` option set at app initialization.
Returns:	A newly initialized Reference.
Return type:	Reference
Raises:	**ValueError** – If the specified path or app is invalid.

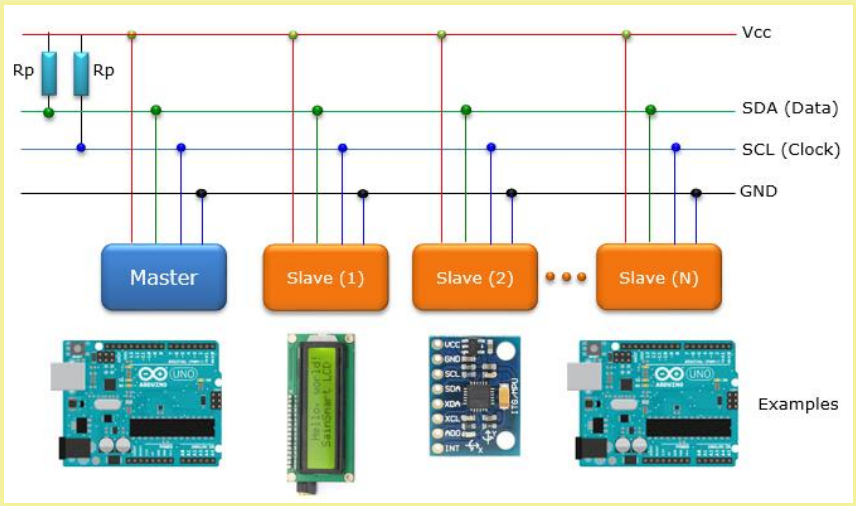

```
########## Firebase Setting ##########
cred = credentials.Certificate("/home/pi/Project/rasberry-596de-firebase-adminsdk-sm1ch-a946597156.json")
firebase_admin.initialize_app(cred,{
    'databaseURL' : 'https://rasberry-596de-default-rtdb.firebaseio.com/'
})
ref = db.reference() #기본 위치 지정
#####################################
```

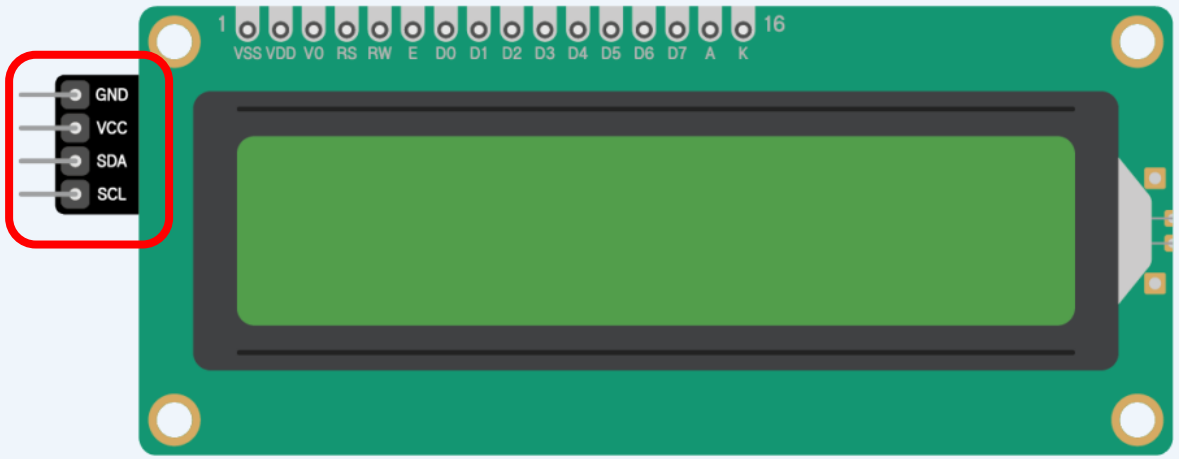
출처: Google Firebase Documentation

외부 라이브러리 – LCD 모듈

```
import RPi_I2C_driver as I2C
```

라이브러리 출처: https://www.recantha.co.uk/blog/?p=4849





출처: 블로그(https://mickael-k.tistory.com/184), 네이버 블로그(https://blog.naver.com/PostView.nhn?blogId=dokkosam&logNo=222285951592&categoryNo=45&parentCategoryNo=0)

SW 구성

핀 설정 / 모터 설정

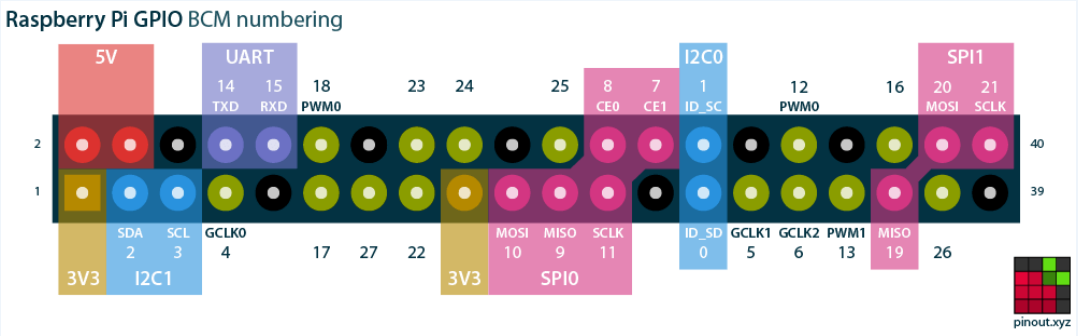
HW

SW

Pin Setting

```
# Define Pins
motor360_pin         = 13    # product output motor
motor180_pin         = 18    # purchase/retrun select motor
siren_pin            = 22    # purchase/retrun select motor
purchase_btn_pin     = 23    # purchase button
return_btn_pin       = 24    # retrun button
call_btn_pin         = 26    # menager call button
sensor500_pin        = 21    # count 500won
sensor100_pin        = 20    # count 100won
sensorStopMotor_pin  = 25    # Sensor to stop 360 degree motor
red_LED_pin          = 27    # red LED
green_LED_pin        = 17    # green LED
```

(BCM 기준)

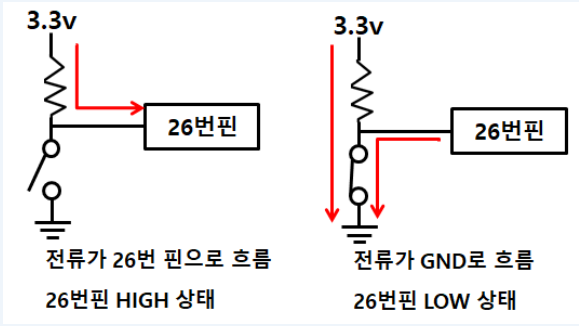


출처: 라즈베리파이 재단 핀맵 (https://pinout.xyz/)

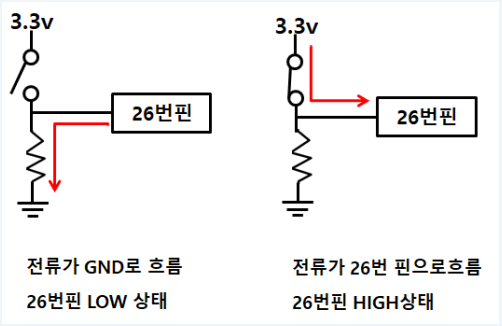
Pin Setting

Broadcom SOC 기준 핀 번호로 세팅
(다른 세팅법: BOARD에 있는 핀 기준으로도 가능)

```
# GPIO Pin Setting
GPIO.setmode(GPIO.BCM)

GPIO.setup(motor360_pin, GPIO.OUT)
GPIO.setup(motor180_pin, GPIO.OUT)
GPIO.setup(siren_pin, GPIO.OUT)
GPIO.setup(purchase_btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(return_btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(call_btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(sensor500_pin, GPIO.IN)
GPIO.setup(sensor100_pin, GPIO.IN)
GPIO.setup(sensorStopMotor_pin, GPIO.IN)
GPIO.setup(red_LED_pin, GPIO.OUT)
GPIO.setup(green_LED_pin, GPIO.OUT)
```



풀업 저항
OFF일때 -> HIGH
ON 일때 -> LOW



풀다운 저항
OFF일때 -> LOW
ON 일때 -> HIGH

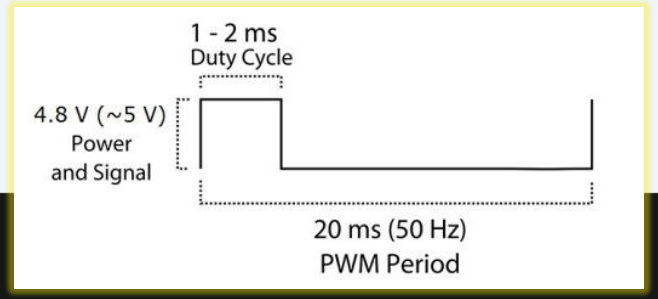
내장 pull-down, pull-up 저항 사용

스위치에 pull-down, pull-up 회로를 만들어주는게 별 거아니지만 귀찮을 때가 많다. 그럴 줄 알고 라즈베리파이 내부에 풀다운/풀업 저항을 만들어 놓고 sw로 활성화 할 수 있도록 되어있다.

```
GPIO.setup(18, GPIO.IN, pull_up_down = GPIO.PUD_UP) # 스위치 안눌렸을 때 on, 눌렸을 때 off
#or
GPIO.setup(18, GPIO.IN, pull_up_down = GPIO.PUD_DOWN) # 스위치 안눌렸을 때 off, 눌렸을 때 on
```

출처: http://lhdangerous.godohosting.com/wiki/index.php/Raspberry_pi_%EC%97%90%EC%84%9C_python%EC%9C%BC%EB%A1%9C_GPIO_%EC%82%AC%EC%9A%A9%ED%95%98%EA%B8%B0
네이버 블로그(https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=simjk98&logNo=221248131170)

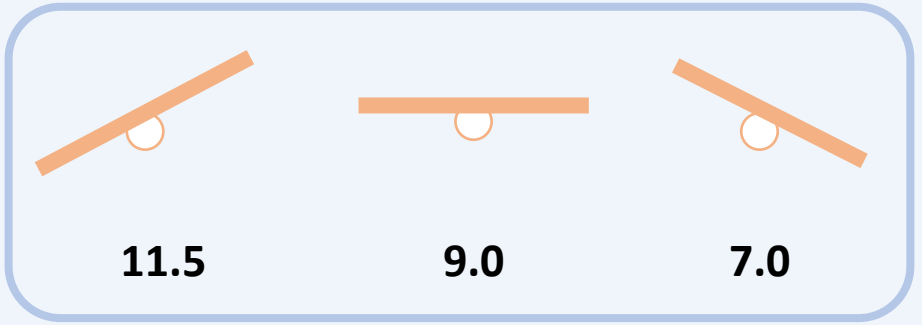
Motor Setting – PWM 설정, 초기화



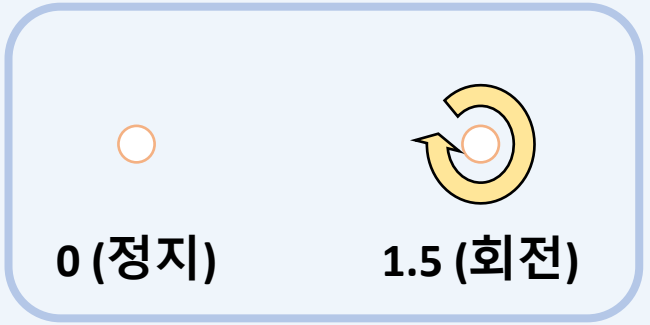

```
# Motor Setting
motor180LeftDutyCycle = 11.5
motor180MiddleDutyCycle = 9.0
motor180RightDutyCycle = 7.0

pwm360 = GPIO.PWM(motor360_pin, 50)  # 50Hz (서보모터 PWM 동작을 위한 주파수)
pwm360.start(0) #서보의 정지 상태
pwm180 = GPIO.PWM(motor180_pin, 50)  # 50Hz (서보모터 PWM 동작을 위한 주파수)
pwm180.start(motor180MiddleDutyCycle) #서보의 0도 위치


# 180도 모터를 가운데로 설정
pwm180.ChangeDutyCycle(motor180MiddleDutyCycle)
time.sleep(1.0)
```

모터가 이동할 시간을 줌

* duty: 1주기 동안 High를 유지하는 기간



180도 서보모터의 주기



360도 서보모터의 주기

SW 구성

VendingMachine 클래스 정의

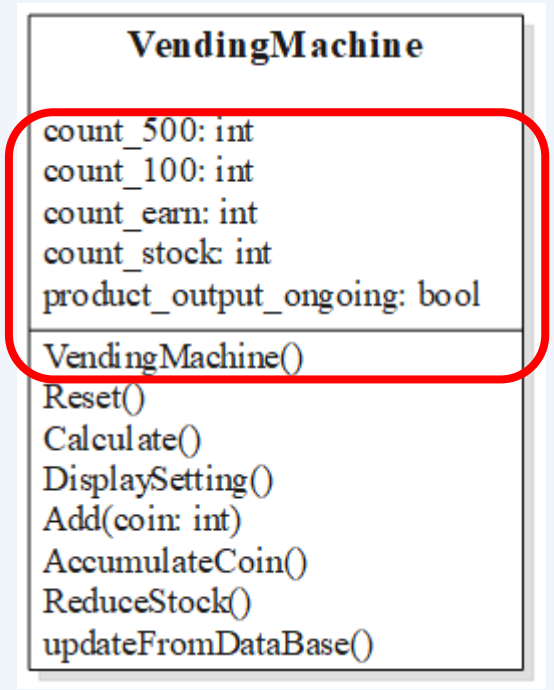
HW

SW

VendingMachine Class

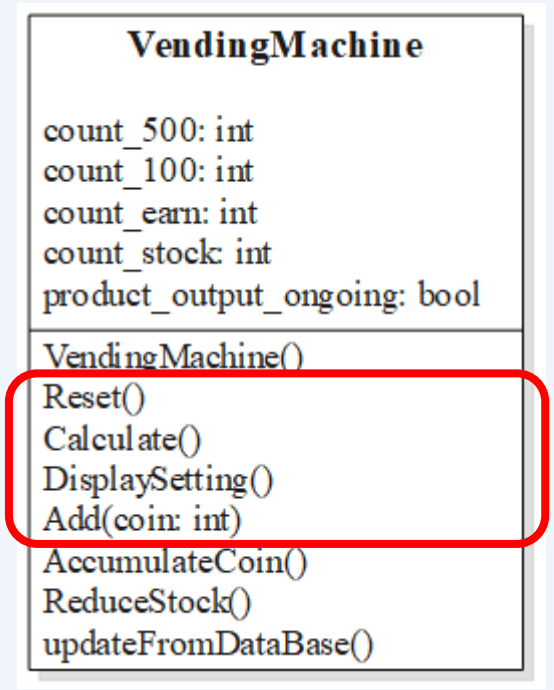
```python
class VendingMachine:
    def __init__(self, count_500, count_100, count_earn, count_stock, product_output_ongoing):
        self.count_500 = count_500
        self.count_100 = count_100
        self.count_earn = count_earn
        self.count_stock = count_stock
        self.product_output_ongoing = product_output_ongoing

    def Reset(self):
        self.count_500 = 0
        self.count_100 = 0

    def Calculate(self):
        return self.count_500 * 500 + self.count_100 * 100

    def DisplaySetting(self):
        mylcd.lcd_display_string(f"Total: {self.Calculate()}     ",2)

    def Add(self, coin):
        if coin == 500:
            self.count_500 += 1
        else:
            self.count_100 += 1

    def AccumulateCoin(self):
        self.count_earn += 700

    def ReduceStock(self):
        self.count_stock -= 1

    def updateFromDatabase(self):
        try:
            self.count_stock = int(db.reference('vendingM/stock').get())
            self.count_earn = int(db.reference('vendingM/earnCoin').get())
            print(f'firestore로부터 데이터를 받아옵니다.')
            print(f'현재 재고: {self.count_stock}개')
            print(f'현재 수입: {self.count_earn}원')

        except:
            print(u'No such document!')
```

VendingMachine Class




```
    def __init__(self, count_500, count_100, count_earn, count_stock, product_output_ongoing):
        self.count_500 = count_500
        self.count_100 = count_100
        self.count_earn = count_earn
        self.count_stock = count_stock
        self.product_output_ongoing = product_output_ongoing
```

VendingMachine Class




```
    def Reset(self):
        self.count_500 = 0
        self.count_100 = 0

    def Calculate(self):
        return self.count_500 * 500 + self.count_100 * 100

    def DisplaySetting(self):
        mylcd.lcd_display_string(f"Total: {self.Calculate()}     ",2)

    def Add(self, coin):
        if coin == 500:
            self.count_500 += 1
        else:
            self.count_100 += 1
```

VendingMachine Class

VendingMachine

count_500: int
count_100: int
count_earn: int
count_stock: int
product_output_ongoing: bool

VendingMachine()
Reset()
Calculate()
DisplaySetting()
Add(coin: int)
AccumulateCoin()
ReduceStock()
updateFromDataBase()

```
https://rasberry-596de-default-rtdb.firebaseio.com/
  vendingM
    Message: ""
    RaspUpdateSignal: 0
    earnCoin: 2100
    stock: 2
```

```python
def AccumulateCoin(self):
    self.count_earn += 700

def ReduceStock(self):
    self.count_stock -= 1

def updateFromDatabase(self):
    try:
        self.count_stock = int(db.reference('vendingM/stock').get())
        self.count_earn = int(db.reference('vendingM/earnCoin').get())
        print(f'firestore로부터 데이터를 받아옵니다.')
        print(f'현재 재고: {self.count_stock}개')
        print(f'현재 수입: {self.count_earn}원')

    except:
        print(u'No such document!')
```

SW 구성

ButtonPushed(), initDisplay() 함수 구현

디스플레이 개체 생성

```
# 디스플레이 개체 생성
mylcd = I2C.lcd()
```

```
class lcd:
   #initializes objects and lcd
   def __init__(self):
      self.lcd_device = i2c_device(ADDRESS)

      self.lcd_write(0x03)
      self.lcd_write(0x03)
      self.lcd_write(0x03)
      self.lcd_write(0x02)

      self.lcd_write(LCD_FUNCTIONSET | LCD_2LINE | LCD_5x8DOTS | LCD_4BITMODE)
      self.lcd_write(LCD_DISPLAYCONTROL | LCD_DISPLAYON)
      self.lcd_write(LCD_CLEARDISPLAY)
      self.lcd_write(LCD_ENTRYMODESET | LCD_ENTRYLEFT)
      sleep(0.2)


   # clocks EN to latch command
   def lcd_strobe(self, data):
      self.lcd_device.write_cmd(data | En | LCD_BACKLIGHT)
      sleep(.0005)
      self.lcd_device.write_cmd(((data & ~En) | LCD_BACKLIGHT))
      sleep(.0001)

   def lcd_write_four_bits(self, data):
      self.lcd_device.write_cmd(data | LCD_BACKLIGHT)
      self.lcd_strobe(data)
```

```
   # write a command to lcd
   def lcd_write(self, cmd, mode=0):
      self.lcd_write_four_bits(mode | (cmd & 0xF0))
      self.lcd_write_four_bits(mode | ((cmd << 4) & 0xF0))

   # write a character to lcd (or character rom) 0x09: backlight | RS=DR<
   # works!
   def lcd_write_char(self, charvalue, mode=1):
      self.lcd_write_four_bits(mode | (charvalue & 0xF0))
      self.lcd_write_four_bits(mode | ((charvalue << 4) & 0xF0))


   # put string function
   def lcd_display_string(self, string, line):
      if line == 1:
         self.lcd_write(0x80)
      if line == 2:
         self.lcd_write(0xC0)
      if line == 3:
         self.lcd_write(0x94)
      if line == 4:
         self.lcd_write(0xD4)

      for char in string:
         self.lcd_write(ord(char), Rs)

   # clear lcd and set to home
   def lcd_clear(self):
      self.lcd_write(LCD_CLEARDISPLAY)
      self.lcd_write(LCD_RETURNHOME)
```

```
   # define backlight on/off (lcd.backlight(1); off= lcd.backlight(0)
   def backlight(self, state): # for state, 1 = on, 0 = off
      if state == 1:
         self.lcd_device.write_cmd(LCD_BACKLIGHT)
      elif state == 0:
         self.lcd_device.write_cmd(LCD_NOBACKLIGHT)

   # add custom characters (0 - 7)
   def lcd_load_custom_chars(self, fontdata):
      self.lcd_write(0x40);
      for char in fontdata:
         for line in char:
            self.lcd_write_char(line)

   # define precise positioning (addition from the forum)
   def lcd_display_string_pos(self, string, line, pos):
    if line == 1:
      pos_new = pos
    elif line == 2:
      pos_new = 0x40 + pos
    elif line == 3:
      pos_new = 0x14 + pos
    elif line == 4:
      pos_new = 0x54 + pos

    self.lcd_write(0x80 + pos_new)

    for char in string:
      self.lcd_write(ord(char), Rs)
```

RPI_I2C_driver 내부의 lcd class

buttonPushed(sentence, m_control1), initDisplay()

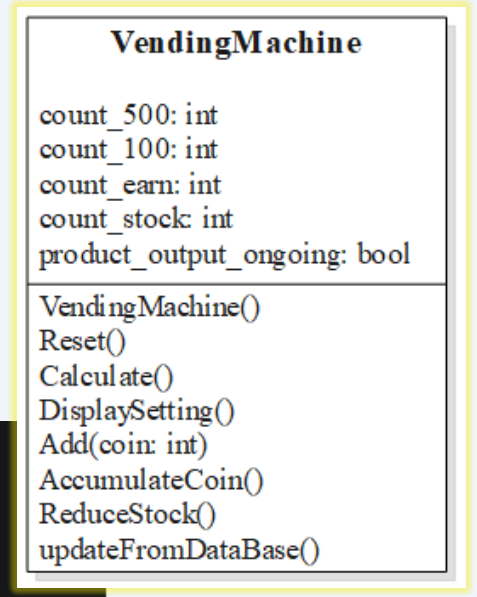
* 버튼이 눌리게 되면, 내부에 담아두고 있던 동전을 반환, 보관하는 동작 중 한 동작을 수행해야 한다.

```
def buttonPushed(sentence, m_control1):
    mylcd.lcd_display_string(f"{sentence}",2)
    pwm180.ChangeDutyCycle(m_control1)
    time.sleep(1.3)
    pwm180.ChangeDutyCycle(motor180MiddleDutyCycle)
    time.sleep(1.0)

def initDisplay(): 디스플레이 초기화 함수
    mylcd.lcd_display_string("Welcome Vend.M  ",1)
    mylcd.lcd_display_string("Insert a coin   ",2)
```

// LCD의 두번째 줄의 문장에 {sentence}를 적음
// 모터의 주기를 m_control1으로 변경 -> 모터의 위치 변경
// 모터가 이동할 시간을 줌
// 모터의 주기를, 모터가 중앙일 때로 변경
// 모터가 이동할 시간을 줌

Initialization

```
# Create Coin & Initialization
vendingMachine = VendingMachine(0, 0, 0, 0, False)
vendingMachine.updateFromDatabase()

# 초기 설정
initDisplay()
```



SW 구성

Interrupt 처리 함수 구현

Callback_By_100

Callback_By_500

Callback_By_return_btn_pin

Callback_By_purchase_btn_pin

Callback_By_call_btn_pin

Callback_By_sensorStopMotorandWarning_btn_pin

Interrupt Functions - Coin

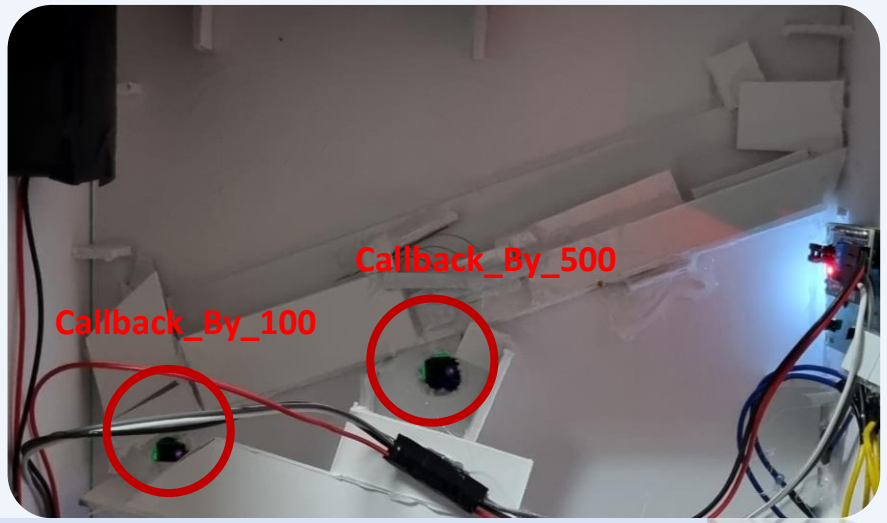
VendingMachine

count_500: int
count_100: int
count_earn: int
count_stock: int
product_output_ongoing: bool

VendingMachine()
Reset()
Calculate()
DisplaySetting()
Add(coin: int)
AccumulateCoin()
ReduceStock()
updateFromDataBase()

```python
def callback_By_500(channel):
    print("500원이 들어옴")
    vendingMachine.Add(500)
    vendingMachine.DisplaySetting() // 현재 넣어준 코인으로 LCD 2번째 줄 변경
```

```python
def callback_By_100(channel):
    print("100원이 들어옴")
    vendingMachine.Add(100)
    vendingMachine.DisplaySetting()
```



HW

SW

Interrupt Functions - return

```
def callback_By_return_btn_pin(channel):
    print("retrun button 눌림")
    vendingMachine.Reset()
    buttonPushed(f"Return Complete", motor180RightDutyCycle)
    initDisplay()
```

// 현재까지 넣은 코인 초기화
// 메시지 표시 + 코인 반환을 위해 모터를 회전
// 디스플레이 초기화

Callback_By_return_btn_pin

VendingMachine

count_500: int
count_100: int
count_earn: int
count_stock: int
product_output_ongoing: bool

VendingMachine()
Reset()
Calculate()
DisplaySetting()
Add(coin: int)
AccumulateCoin()
ReduceStock()
updateFromDataBase()

HW

SW

Interrupt Functions - purchase

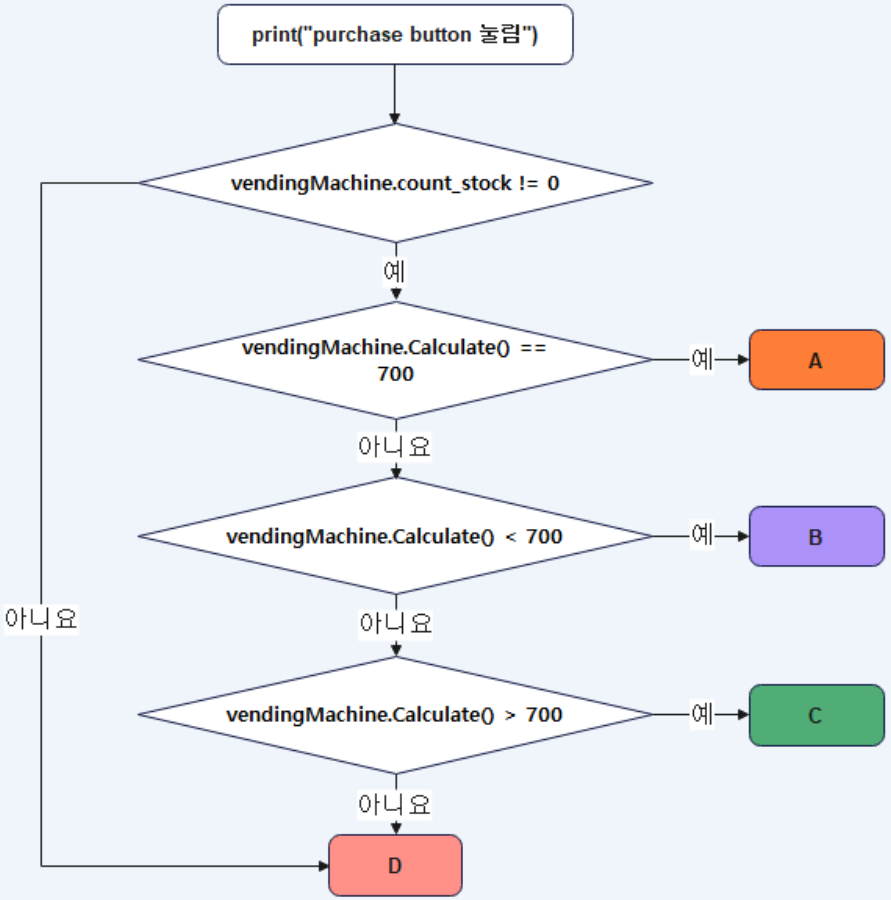
```
def callback_By_purchase_btn_pin(channel):
    print("purchase button 눌림")

    if vendingMachine.count_stock != 0:
        if vendingMachine.Calculate() == 700:
            print("700원 충족 - 구매됨")
            GPIO.output(green_LED_pin, 1)
            buttonPushed(f"Purchase Success", motor180LeftDutyCycle)
            vendingMachine.AccumulateCoin()
            vendingMachine.ReduceStock()
            vendingMachine.Reset()

            db.reference('vendingM').update({'earnCoin':vendingMachine.count_earn})
            db.reference('vendingM').update({'stock':vendingMachine.count_stock})
            print(f"총 수익:{vendingMachine.count_earn} - firebase에 업로드 완료")
            print(f"남은 재고:{vendingMachine.count_stock} - firebase에 업로드 완료")

            vendingMachine.product_output_ongoing = True
            # product out
            pwm360.ChangeDutyCycle(1.5)
            # 이 뒤로는 sensorStopMotor의 Interrupt로 처리
```

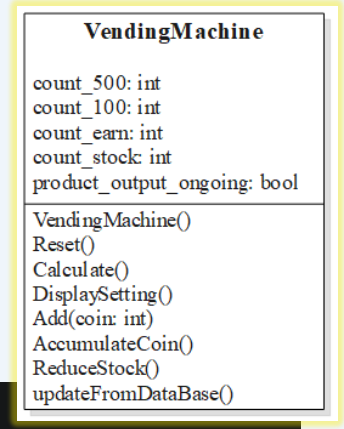

```
        elif vendingMachine.Calculate() < 700:
            print("700원 충족되지 못함 - 구매 불가")

            mylcd.lcd_display_string(f"Can't purchase  ",1)
            mylcd.lcd_display_string(f"Not Enough coin",2)
            time.sleep(1.5)
            initDisplay()
            vendingMachine.DisplaySetting()

        elif vendingMachine.Calculate() > 700:
            print("700원 충족되지 못함 - 구매 불가")

            mylcd.lcd_display_string(f"Can't purchase  ",1)
            buttonPushed(f"Too many coins", motor180RightDutyCycle)
            vendingMachine.Reset()
            time.sleep(1.5)

            initDisplay()
    else:
        print("재고가 없음 -  구매 불가")
        mylcd.lcd_display_string(f"Can't purchase  ",1)
        buttonPushed(f"Out of stock    ", motor180RightDutyCycle)
        time.sleep(1.5)
        vendingMachine.Reset()
        initDisplay()
```

HW

SW

Interrupt Functions - purchase





A

```
print("700원 충족 - 구매됨")
GPIO.output(green_LED_pin, 1)
buttonPushed(f"Purchase Success", motor180LeftDutyCycle)
vendingMachine.AccumulateCoin()
vendingMachine.ReduceStock()
vendingMachine.Reset()

db.reference('vendingM').update({'earnCoin':vendingMachine.count_earn})
db.reference('vendingM').update({'stock':vendingMachine.count_stock})
print(f"총 수익:{vendingMachine.count_earn} - firebase에 업로드 완료")
print(f"남은 재고:{vendingMachine.count_stock} - firebase에 업로드 완료")

vendingMachine.product_output_ongoing = True
# product out
pwm360.ChangeDutyCycle(1.5)
# 이 뒤로는 sensorStopMotor의 Interrupt로 처리
```

Interrupt Functions - purchase

VendingMachine

count_500: int
count_100: int
count_earn: int
count_stock: int
product_output_ongoing: bool

VendingMachine()
Reset()
Calculate()
DisplaySetting()
Add(coin: int)
AccumulateCoin()
ReduceStock()
updateFromDataBase()

A

```
print("700원 충족 - 구매됨")
GPIO.output(green_LED_pin, 1)                                  // 초록 LED에 불이 들어옴
buttonPushed(f"Purchase Success", motor180LeftDutyCycle)       // "Purchase Success"를 LCD에 표시 + 동전을 보관
vendingMachine.AccumulateCoin()                                // 현재까지 번 돈에 +700
vendingMachine.ReduceStock()                                   // 재고를 1 감소
vendingMachine.Reset()                                         // 현재까지 들어온 500 / 100을 0으로 초기화

db.reference('vendingM').update({'earnCoin':vendingMachine.count_earn})
db.reference('vendingM').update({'stock':vendingMachine.count_stock})
print(f"총 수익:{vendingMachine.count_earn} - firebase에 업로드 완료")
print(f"남은 재고:{vendingMachine.count_stock} - firebase에 업로드 완료")

vendingMachine.product_output_ongoing = True  // 출력 중임을 보이는 변수
# product out
pwm360.ChangeDutyCycle(1.5)                   // 상품 출력
# 이 뒤로는 sensorStopMotor의 Interrupt로 처리
```

출력 중인 모터를 멈추는 때는 senseorStopMotor의 Interrupt가 발생했을 때

실시간 데이터베이스

데이터 규칙 백업 사용량

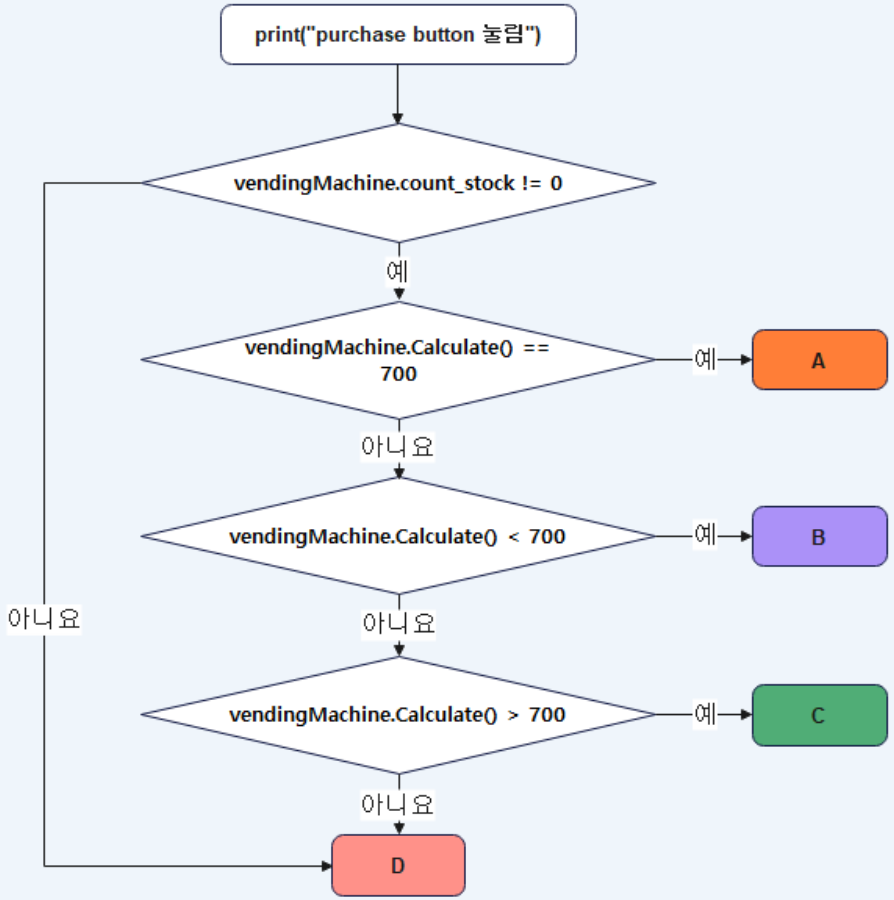
https://rasberry-596de-default-rtdb.firebaseio.com

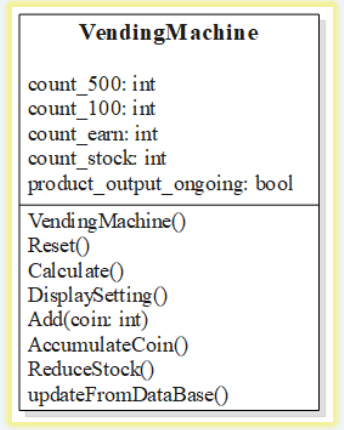
https://rasberry-596de-default-rtdb.firebaseio.com/
- vendingM
  - Message: ""
  - RaspUpdateSignal: 0
  - earnCoin: 2100
  - stock: 2

HW

SW

Interrupt Functions - purchase





B

```
print("700원 충족되지 못함 - 구매 불가")

mylcd.lcd_display_string(f"Can't purchase  ",1)
mylcd.lcd_display_string(f"Not Enough coin",2)
time.sleep(1.5)
initDisplay()
vendingMachine.DisplaySetting()
```

디스플레이

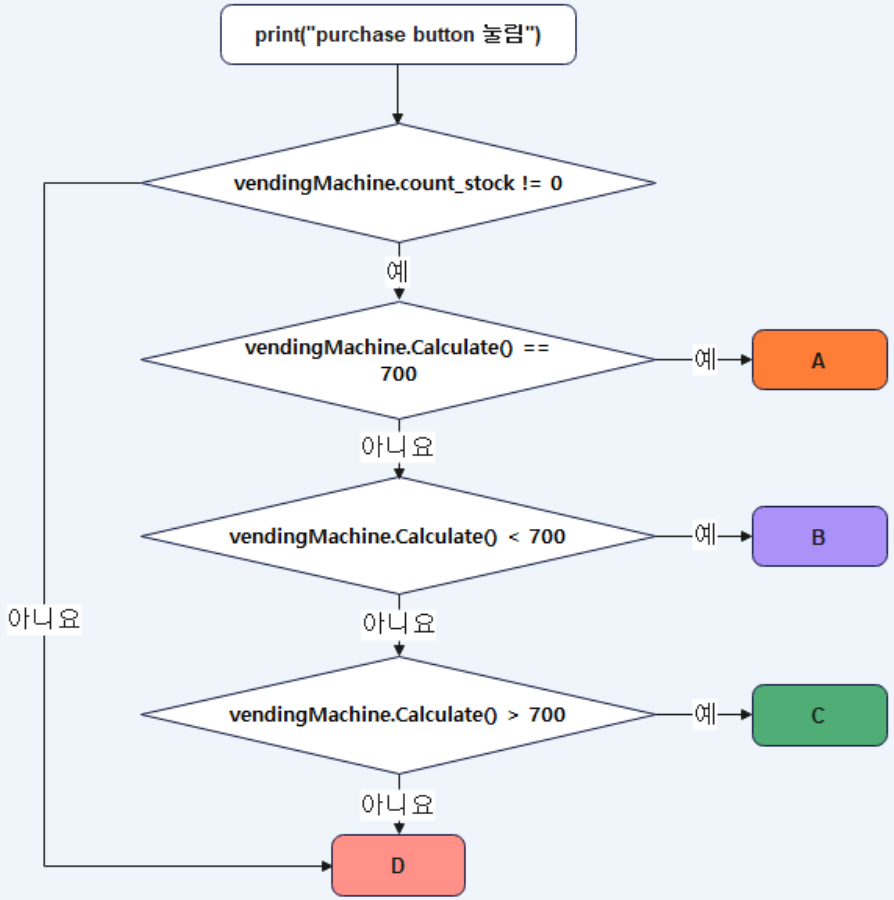
초기화 후 지금까지 넣은 코인 다시 표시

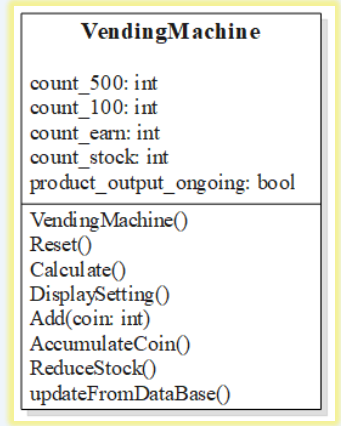
C

```
print("700원 충족되지 못함 - 구매 불가")

mylcd.lcd_display_string(f"Can't purchase  ",1)
buttonPushed(f"Too many coins", motor180RightDutyCycle)
vendingMachine.Reset()
time.sleep(1.5)

initDisplay()
```

// 현재까지 넣은 코인 초기화
// 메시지 표시 + 코인 반환을 위해 모터를 회전
// 현재까지 들어온 500 / 100을 0으로 초기화
// 디스플레이 초기화

Interrupt Functions - purchase





D

```
print("재고가 없음 -  구매 불가")
mylcd.lcd_display_string(f"Can't purchase  ",1)
buttonPushed(f"Out of stock    ", motor180RightDutyCycle)
time.sleep(1.5)
vendingMachine.Reset()
initDisplay()
```

// 현재까지 넣은 코인 초기화
// 메시지 표시 + 코인 반환을 위해 모터를 회전
// 현재까지 들어온 500 / 100을 0으로 초기화
// 디스플레이 초기화

=> c번과 메시지만 다르고 동일한 기능

Interrupt Functions - call

```
def callback_By_call_btn_pin(channel):
    db.reference('vendingM').update({'Message':"자판기의 사용자에게 문제가 발생하였습니다."})
    print("call button 눌림 - 관리자에게 알림")
    mylcd.lcd_display_string(f"Call Success!   ",1)
    mylcd.lcd_display_string(f"Wait a minute.  ",2)
    time.sleep(1.5)
    initDisplay()
```

VendingMachine

count_500: int
count_100: int
count_earn: int
count_stock: int
product_output_ongoing: bool

VendingMachine()
Reset()
Calculate()
DisplaySetting()
Add(coin: int)
AccumulateCoin()
ReduceStock()
updateFromDataBase()

vendingM
- Message: "자판기의 사용자에게 문제가 발생하였습니다."
- RaspUpdateSignal: 0
- earnCoin: 3500
- stock: 2

HW

SW

Interrupt Functions – Stop motor and Warning

```
def callback_By_sensorStopMotorandWarning_pin(channel):
    if vendingMachine.product_output_ongoing == True: 상품이 출력 중일 때
        print("과자가 나옴")
        pwm360.start(0) #서보의 정지 상태
        GPIO.output(green_LED_pin, 0)
        initDisplay()
        time.sleep(8.0)
        vendingMachine.product_output_ongoing = False

    else: 상품이 출력 중이지 않을 때
        GPIO.output(siren_pin, 1)
        GPIO.output(red_LED_pin, 1)
        time.sleep(1.0)
        GPIO.output(siren_pin, 0)
        GPIO.output(red_LED_pin, 0)

        db.reference('vendingM').update({'Message':"자판기의 도난이 의심됨"})
        print(f"자판기의 도난이 의심됨 - 관리자에게 알림")
```

// 회전하던 모터를 정지 (Callback_By_purchase_btn_pin()에서 모터를 작동했었음)

// 초록 LED 끄기 (Callback_By_purchase_btn_pin()에서 LED를 켰었음)

// 상품 출력 변수를 출력 중이지 않은 상태로 변경

레드 LED와 사이렌 부저를 켰다 끄기

// 관리자에게 알림

```
vendingM
    Message: "자판기의 도난이 의심됨"
    RaspUpdateSignal: 0
    earnCoin: 3500
    stock: 2
```

VendingMachine

count_500: int
count_100: int
count_earn: int
count_stock: int
product_output_ongoing: bool

VendingMachine()
Reset()
Calculate()
DisplaySetting()
Add(coin: int)
AccumulateCoin()
ReduceStock()
updateFromDataBase()

Callback_By_sensorStopMotorandWarning_btn_pin

Interrupt Setting

```
GPIO.add_event_detect(sensor100_pin, GPIO.FALLING, callback=callback_By_100, bouncetime=250)
GPIO.add_event_detect(sensor500_pin, GPIO.FALLING, callback=callback_By_500, bouncetime=250)
GPIO.add_event_detect(return_btn_pin, GPIO.FALLING, callback=callback_By_return_btn_pin, bouncetime=5000)
GPIO.add_event_detect(purchase_btn_pin, GPIO.FALLING, callback=callback_By_purchase_btn_pin, bouncetime=5000)
GPIO.add_event_detect(call_btn_pin, GPIO.FALLING, callback=callback_By_call_btn_pin, bouncetime=5000)
GPIO.add_event_detect(sensorStopMotor_pin, GPIO.FALLING, callback=callback_By_sensorStopMotorandWarning_pin, bouncetime=5000)
```

함수명	기능
wait_for_edge(핀번호, 상태설정)	에지가 발생할때까지 프로그램을 수행하지 않고 대기함. 상태설정 : GPIO.RISING : 0에서 1로 전이시 이벤트 발생 GPIO.FALLING: 1에서 0으로 전이시 이벤트 발생 GPIO.BOTH : 상태변화가 생기면 이벤트 발생
event_detected(핀번호)	폴링 방식과 마찬가지로 루프내에서 사용되며, 폴링과 다른 점은 이벤트가 발생한 것을 누락하지 않고 감지함. 폴링방식과 인터럽트 방식이 혼합된 형태. 사전에 add_event_detect() 함수를 호출하여 이벤트를 등록 한 후 event_detected() 함수를 호출하면 해당 이벤트가 발생하였는지에 따라 True 또는 False를 반환.
add_event_detect(핀번호, 상태설정, callback='함수명', bouncetime=숫자)	위의 두가지 방식은 어찌보면 엄밀한 의미의 인터럽트라고 보기는 어려울 수 있다. CPU의 제어권을 넘겨 받아 실행을 수행하는 방식은 아니기 때문이다. add_event_detect함수에서 callback에 수행될 함수를 지정하면 해당 이벤트가 발생될 경우 쓰레드를 생성하여 callback 함수를 실행하게 된다. bouncetime 옵션은 버튼이 눌려질때 두번이상 callback 함수를 호출하는 것(스위치 바운스)을 방지하기 위하여 일정기간의 에지를 무시하도록 설정하는 옵션으로 단위는 밀리세컨드 이다.

Pin을 pull_up으로 설정하여 default 상태가 1

```
# GPIO Pin Setting
GPIO.setmode(GPIO.BCM)

GPIO.setup(motor360_pin, GPIO.OUT)
GPIO.setup(motor180_pin, GPIO.OUT)
GPIO.setup(siren_pin, GPIO.OUT)
GPIO.setup(purchase_btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(return_btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(call_btn_pin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(sensor500_pin, GPIO.IN)
GPIO.setup(sensor100_pin, GPIO.IN)
GPIO.setup(sensorStopMotor_pin, GPIO.IN)
GPIO.setup(red_LED_pin, GPIO.OUT)
GPIO.setup(green_LED_pin, GPIO.OUT)
```

출처: 네이버 블로그(https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=simjk98&logNo=221248131170)

SW 구성

main 함수 구현

Main

```python
# Polling Function (Main)
try:
    while True:
        print("대기중")
        time.sleep(2)

        if int(db.reference('vendingM/RaspUpdateSignal').get()) == 1:
            vendingMachine.updateFromDatabase()
            db.reference('vendingM').update({'Message':"자판기 정보 업데이트 완료"})
            db.reference('vendingM').update({'RaspUpdateSignal':0})

# 프로그램 종료, (Ctrl + C를 입력)
except KeyboardInterrupt:
    print('프로그램을 종료합니다.')

finally:
    pwm360.stop()
    pwm180.stop()
    GPIO.cleanup()
```

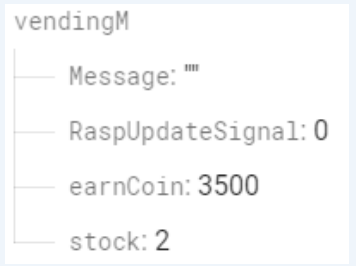
→ Exception 처리

→ 자원 반환

Main

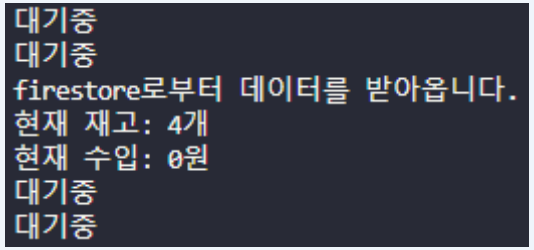
```
if int(db.reference('vendingM/RaspUpdateSignal').get()) == 1:
    vendingMachine.updateFromDatabase()
    db.reference('vendingM').update({'Message':"자판기 정보 업데이트 완료"})
    db.reference('vendingM').update({'RaspUpdateSignal':0})
```

자판기 작동 중, 파이어베이스로부터 정보 업데이트 매커니즘



원하는 정보 수정: earnCoin, stock

RaspUpdateSignal에 1을 set



자판기 업데이트

RaspUpdateSignal에 0으로 reset

작동 영상

임베디드시스템 기말 프로젝트

IOT 자판기

32182479 안선홍

감사합니다.