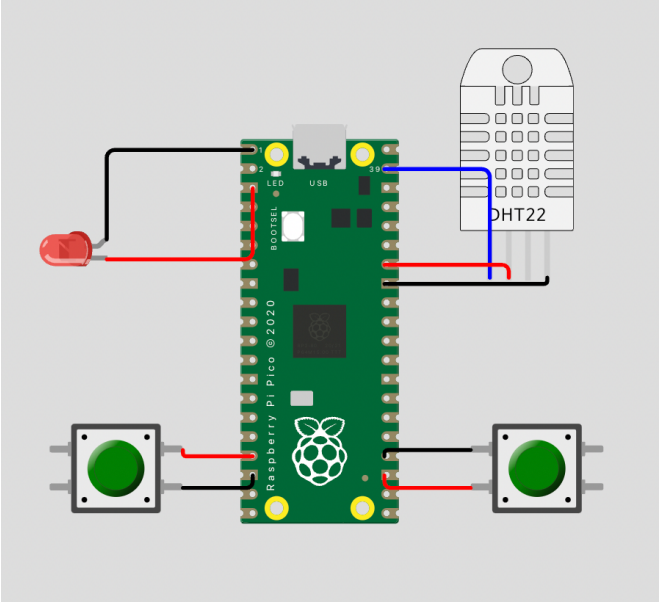
컴퓨터시스템과피지컬컴퓨팅 기말과제 P-2 보고서

연세대학교 학부대학 상경계열 경제학과

2024121183 최병준

1. 전체 센서 구성도



led1
- A : GP0
- C : GND.1

Pushbutton-1 (주기 전환 버튼)
- 1.r : GP13
- 2.r : GND.4

Pushbutton-2 (출력 버튼)
- 1.l : GP18
- 2.l : GND.5

DHT22
- VCC : VSYS
- SDA : GP28
- GND : GND.7

2. 작동 코드의 핵심 아이디어

전체적인 코드는 while 문을 이용하여 프로그램이 실행되는 내내, 마지막으로 온습도를 측정한 시각과 현재 시각의 차이가 설정된 T초 주기보다 크거나 같을 경우 온습도를 측정, 즉 T초 주기마다 온습도를 측정하는 메인 루프를 중심으로 이루어져 있다. Pushbutton을 누르면 interrupt handler를 통해 주기를 변경하거나 최근 10개의 측정값을 출력하는 callback 함수가 실행된다. 시스템 실행 후 지금까지 소요된 시간을 밀리초로 나타내는 time.ticks_ms() 와 밀리초 간의 차이를 계산해주는 time.ticks_diff() 함수를 활용해 time.sleep() 와 같은 blocking 방식이 아닌 non-blocking 방식을 활용하였고, print_flag 라는 변수를 선언해 출력 버튼이 눌릴 경우 while loop 안에서 print_flag 의 boolean 값을 판단하여 측정값을 출력하는 방식을 채택, 출력 버튼의 interrupt handler 동작 시간을 최소화하여 코드의 정확도를 제고하였다.

3. 코드 해석 및 논리적 흐름

1) 사전 준비

우선 LED, Pushbutton, DHT22 센서를 사용하기 위해 dht, Pin 라이브러리를 import하고, 코드의 작동을 제어하기 위해 time 라이브러리 또한 import한다.

```
1  import time
2  import dht
3  from machine import Pin
4
5  file = open('./input.txt', 'r', encoding='EUC-KR')
6
7  for line in file:
8    period = line.strip().split(' ')
9    period = [int(i) for i in period]
```

주기가 기록된 input.txt를 open() 함수를 통해 열고, input.txt의 각 줄을 strip() 함수를 이용해 줄바꿈 문자(₩n)을 제거한 뒤 각 주기가 띄어쓰기로 분리되어 있기 때문에 split() 함수를 사용하여 input.txt 내 저장된 주기를 담는 period 리스트를 만든다. 또한 list comprehension을 이용해 string으로 저장된 각 주기를 int로 바꾼다.

```
    period_index = 0
    humidity_data = []
    temperature_data = []
    last_measure_time = time.ticks_ms()
    print_flag = False
```

```
{
  "type": "wokwi-pushbutton",
  "id": "btn1",
  "top": 150.2,
  "left": -96,
  "attrs": { "bounce": "0", "color": "green" }
},
{
  "type": "wokwi-pushbutton",
  "id": "btn2",
  "top": 150.2,
  "left": 115.2,
  "attrs": { "bounce": "0", "color": "green" }
},
```

다음으로 period 리스트 내에서 현재 지정된 주기 T를 가리키기 위한 period_index 변수를 선언하고, 측정한 온습도 데이터를 담을 리스트를 선언한 뒤 출력 버튼 기능 구현을 위한 print_flag 또한 선언한다. 출력 버튼의 세부적인 작동 설명은 아래에서 설명하였다. 프로그램 작동 시각을 기록하기 위해 time.ticks_ms() 함수를 사용해 last_measure_time를 지정한다. 이 함수는 시스템이 시작된 이후의 밀리초를 반환하는 값으로, T초가 지났는지 여부를 판단하는 기준이 된다.

또한 전압이 0V와 3.3V 사이에서 변동하는 과정에서 불완전한 전류의 흐름이 발생하는 Bouncing 현상을 예방하기 위해, diagram.json에서 "bounce" : "0" attribute를 추가했다.

2) GPIO, 센서, main.py 연동

Raspberry Pi Pico에는 입출력을 담당하는 GPIO가 존재한다. GPIO 내 GP를 사용해 입출력을 받거나, GND를 사용해 음극과 연결한다. LED를 연결할 경우, 양극은 GP, 음극은 GND와 연결해 양극에 0(0V)을 주거나 1(3.3V)을 줘 LED를 키거나 끌 수 있다. Pushbutton의 경우 두 개의 핀을 각각 GP와 GND에 연결하며, PULL_MODE에 따라 그 동작이 달라진다. PULL_UP 모드의 경우, 기본 상태가 1(3.3V)가 되며 버튼이 눌리면 0(0V)가 된다. 이때 버튼의 pin 한 쪽은 GND에 연결되어야 한다. PULL_DOWN의 경우 반대로 기본 상태가 0(0V)가 되며, 버튼이 눌릴 경우 1(3.3V)가 된다. 이때 버튼의 pin 한 쪽은 power(GPIO의 VSYS, 3V3 등)와 연결되어야 한다. DHT22 센서의 경우, 총 핀이 4개가 존재하는데, 전력을 담당하는 VCC는 power와, 센서 데이터를 GPIO로 전송하는 SDA의 경우 GP에, DHT22의 GND는 GPIO의 GND에 연결되어야 한다. 나머지 하나의 핀은 NC인데, 사용하지 않는 핀이다.

```
    period_button = Pin(13, Pin.IN, Pin.PULL_UP)
    print_button = Pin(18, Pin.IN, Pin.PULL_UP)
    sensor = dht.DHT22(Pin(28))
    led = Pin(0, Pin.OUT)
```

왼쪽 코드에서는 Raspberry Pi Pico와 센서, 메인 코드를 연결한다. led1은 GP0과 GND1, 주기를 변경하는 버튼(Pushbutton-1)은 GP13과 GND4, 출력 버튼(Pushbutton-2)은 GP18과 GND5, DHT22의 VCC는 pico에서 전력을 수급하기 위해 VSYS와, 데이터를 전송하는 SDA는 pico의 GP28, GND는 GND7과 연결하였다.

작성한 코드에서는 전선 연결의 편의성을 위해 Pushbutton의 PULL_MODE를 PULL_UP 모드로 사용하였다. DHT22의 경우 데이터 처리 과정이 복잡하지만, dht 모듈을 사용한다면 처리 과정을 간단하게 만들 수 있다. dht.DHT22() 함수를 사용해 DHT22로부터 데이터가 들어오는 GP28을 할당하였고, led는 GP0과 연결하였다. Pin.IN과 Pin.OUT은 각각 GPIO의 GP가 출력을 담당하는지 입력을 담당하는지를 의미하며, led는 GPIO에서 출력하는 값을 바탕으로 작동하므로 Pin.OUT을, Pushbutton 2개는 버튼의 출력값이 GPIO로 들어오므로 Pin.IN을 설정하였다.

3) Pushbutton-1, Pushbutton-2 (주기 전환 버튼, 출력 버튼) 의 구현

두 버튼이 눌릴 경우, 그 즉시 기능이 실현되어야 하기 때문에 interrupt handler를 활용해야 한다. irq() 함수를 통해 interrupt handler의 방식과 그 callback 함수를 설정할 수 있다. interrupt handler의 방식은 두 가지로 나뉘는데, IRQ_FALLING과 IRQ_RISING이 그것이다. IRQ_FALLING은 장치의 신호가 1에서 0으로 가면 callback 함수를, IRQ_RISING은 0에서 1로 가면 callback 함수를 호출하는 방식이다. 앞서 Pushbutton을 PULL_UP 모드로 설정했으므로, 그 기본값이 1이며 버튼이 눌리면 0으로 신호가 변화하므로, IRQ_FALLING을 사용하는 것이 적절하다.

```
22  def period_button_callback(channel):
23    global period_index
24    period_index += 1
25
26  period_button.irq(period_button_callback, Pin.IRQ_FALLING)
```

주기 전환 버튼(Pushbutton-1)의 경우, 앞서 선언한 전역변수 period_index를 함수 내에서 사용하기 위해 global 키워드를 사용하고, period_index를 하나씩 증가시키는 callback 함수를 구성하였다. 뒤에서 설명하겠지만, T1, T2, T3를 period 리스트에 담았으니 period_index를 0에서 2 사이의 값으로 설정하는 방법도 있다. 하지만 period_index를 계속해서 증가시키며 리스트 index로 period_index를 사용할 때에는 (period_index % 3)과 같은 형태로 인덱스를 지정해 모듈로 연산을 활용하는 방법이 더 코드 작성에 용이하기 때문에 해당 방법을 채택하였다.

```
28  def print_button_callback(channel):
29    global print_flag
30    print_flag = True
31
32  print_button.irq(print_button_callback, Pin.IRQ_FALLING)
```

출력 버튼(Pushbutton-2)의 경우, interrupt가 발생했을 때 지연이 발생해 측정 과정이 미뤄지는 것을 방지하기 위해, flag를 활용하였다. 전역변수 print_flag를 기본값 False로 선언한 뒤, 출력 버튼이 눌릴 경우 print_flag를 True로 변경하고, while 루프 내에서 print_flag가 True일 경우 최근 10개의 측정값을 출력하는 코드를 구성하였다. 이렇게 코드를 구성한 결과, interrupt handler 내에 print문을 작성했을 때보다 측정 과정이 밀리는 경우가 크게 줄어들었다. wokwi 내에서 실행할 때, 주기가 1초라고 가정하면 특정 시각의 초와 측정 횟수가 일치해야 한다(10.2초에 출력 버튼을 누를 경우 측정 횟수도 10회가 되어야 한다. 1초, 2초, 3초 … 9초, 10초에 측정한 값이 저장되기 때문). print문이 interrupt handler 내에 있을 경우 일치하지 않는 경우가 발생했는데, flag를 사용할 경우 이러한 문제점이 해결되는 것을 관찰하였다.

```
50    if print_flag:
51      print(f"측정 횟수: {len(temperature_data)}")
52      print(f"온도 평균/최대: {(sum(temperature_data[-
53      print(f"습도 평균/최대: {(sum(humidity_data[-10:
54      print_flag = False
55    time.sleep_ms(10)
```

왼쪽 코드는 메인 루프 내에서 출력 버튼의 기능을 구현한 코드이다. 만약 print_flag가 True, 즉 출력 버튼인 Pushbutton-2가 눌렸을 경우, f-string을 이용해 관련 정보를 출력한다. 측정 횟수는 곧 temperature_data나 humidity_data 리스트의 길이값이므로 이를 출력하고, 온습도의 평균과 최대는 최근 10개 값만을 대상으로 구하기 때문에, list slicing을 활용해 리스트의 뒤에서 10개의 값만을 대상으로 평균과 최댓값을 출력하도록 작성하였다. 또한 소수점 자릿수 포맷팅을 위해 :.2f 를 붙여 소수점 2자리수까지 출력하였다. 출력이 완료되었으면 print_flag는 False로 변경된다.

4) 온습도 측정 기능 구현

```
34  while True:
35    current_time = time.ticks_ms()
36    time_diff = time.ticks_diff(current_time, last_measure_time)
37
38    if time_diff >= period[period_index % 3] * 1000:
39      led.value(1)
40      sensor.measure()
41      temp = sensor.temperature()
42      hum = sensor.humidity()
43      temperature_data.append(temp)
44      humidity_data.append(hum)
45      last_measure_time = current_time
```

우선 현재의 시각을 time.ticks_ms() 함수를 통해 current_time 변수에 담아 저장한다. 그 후 time_diff 변수에 두 시각의 차이를 계산해주는 time.ticks_diff() 함수를 활용해 현재의 시각과 마지막으로 온습도를 측정한 시각의 차이를 구한다. 단순하게 time_diff = current_time – last_measure_time 으로 구할 경우 overflow가 발생할 수 있기 때문에, time.ticks_diff() 함수를 사용하는 것이 안전하다. 만약 두 시각의 차이인 time_diff가 설정된 주기보다 같거나 클 경우, 즉 측정해야 하는 시점일 경우 led와 연결된 GP0의 출력을 1로 해 led를 킨 뒤, DHT22 센서로부터 온습도를 측정한다. time.ticks_diff() 함수는 밀리초 단위로 값을 리턴하기 때문에, 설정된 초 단위의 주기를 밀리초로 바꿔주는 작업을 시행하기 위해 1000을 곱해주었다. 이렇게 되면 측정이 이루어진 시각부터의 시간(time_diff)가 측정 주기보다 크거나 같게 되면, 즉 측정할 주기가 다시 찾아오면 측정이 이뤄지는 구조인 것이다. DHT22와 연동된 변수인 sensor의 measure() 함수는 아날로그 방식으로 수집된 온습도 정보를 디지털로 변환하는 전처리 과정을 수행해준다. 이를 통해 단순히 sensor.temperature(), sensor.humidity() 를 참조하

는 것만으로도 아날로그로 측정된 온습도를 디지털로 활용할 수 있게 된다. 앞서 선언한 온습도 정보를 담을 리스트인 temperature_data, humidity_data에 모든 정보를 append 해 저장한다. 또한 측정이 수행되었으므로 last_measure_time을 current_time으로 바꿔준다.

5) LED 깜빡임 구현

```
39    led.value(1)

47    if time_diff >= period[period_index % 3] * 1000 / 2:
48      led.value(0)
```

또한 LED의 깜빡임을 표현하기 위해, 측정이 이뤄지는 if문 내에서 LED가 연결된 GP0의 출력을 1로 바꿔 LED를 켜고, 주기의 절반 정도가 지난 시점에서 GP0의 출력을 다시 0으로 바꿔 꺼지는 기능을 구현해, 주기의 절반 정도 동안 led가 켜지는 기능을 구현하였다.

6) 마무리

```
55    time.sleep_ms(10)
```

이때 while 문을 제어하는 코드가 interrupt handler 밖에 없어 interrupt handler가 동작하지 않으면 while문의 코드는 무한히 반복되고, 이는 CPU에 큰 부하를 준다. 따라서 time.sleep_ms()을 통해 매 반복문마다 10ms 지연을 둬 CPU의 부하를 줄이면서도 전체적인 코드 동작에 큰 영향이 가지 않도록 한다. 100번 정도 반복문이 반복되면 1초의 지연이 생기는 것이므로, 장기적으로 볼 때에는 바람직하지 않은 방법이므로 CPU 성능에 맞춰 지연 시간을 줄이는 방안을 고려해볼 수 있다.

4. 결론

처음 과제 안내사항을 읽을 때, 단순히 time.sleep()을 사용한다면 쉽게 코드를 짤 수 있을 것이라고 생각했지만, interrupt handler를 사용하면서 그 처리 과정이 지연되게 되었고, 측정 횟수가 의도했던 횟수보다 더 적게 나오는 등의 문제점이 발생하였다. 이를 해결하기 위해, 프로그램 전체가 time.sleep()으로 멈추는 것이 아닌, 프로그램 내에서 시간을 직접 계산해 설정된 주기와 비교하여 프로그램을 멈추지 않고도 동시에 온습도 측정, 주기 변환, 측정값 출력 기능을 동시에 진행할 수 있는 새로운 코드를 짜게 되었다. 버튼이 눌렸을 경우 즉각 기능이 실행되도록 interrupt handler을 활용했으며, 10개의 측정값만을 저장하는 복잡한 코드 없이 list slicing을 통해 list에 저장된 값 중 뒤에서 10개, 즉 가장 최근에 저장된 값 10개를 불러오는 코드도 활용하였다. 다만, CPU 과부화를 막기 위해 넣은 마지막 줄의 time.sleep_ms(10) 코드는 장기적으로 봤을 때 오차를 일으킬 가능성이 있으므로, CPU 성능에 따라 밀리초를 점점 줄일 필요성이 있다.