

# REPORT

## 전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: <http://www.ieee.org>)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임:** 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제:** 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성:** 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지:** 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해:** 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성:** 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세:** 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기:** 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공정하게 대한다.
9. **도덕성:** 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애:** 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

<융합캡스톤디자인 최종보고서>

학 부: 전자공학과

제출일: 2023. 06. 25

과목명: 융합캡스톤디자인1

교수명: 안효복 교수님

분 반: C018-3

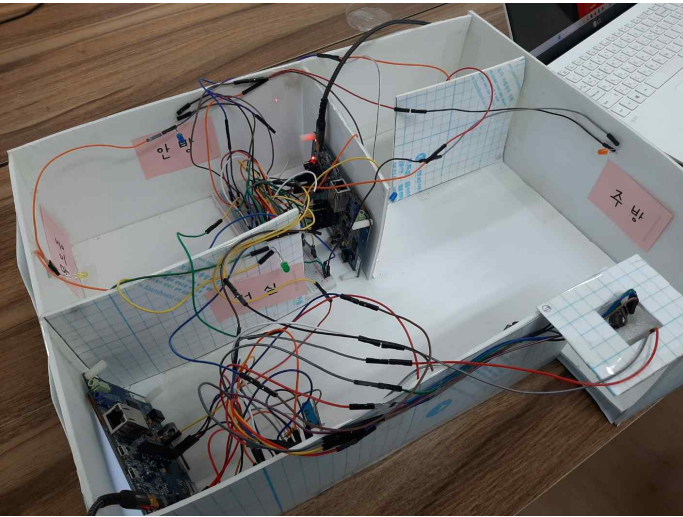

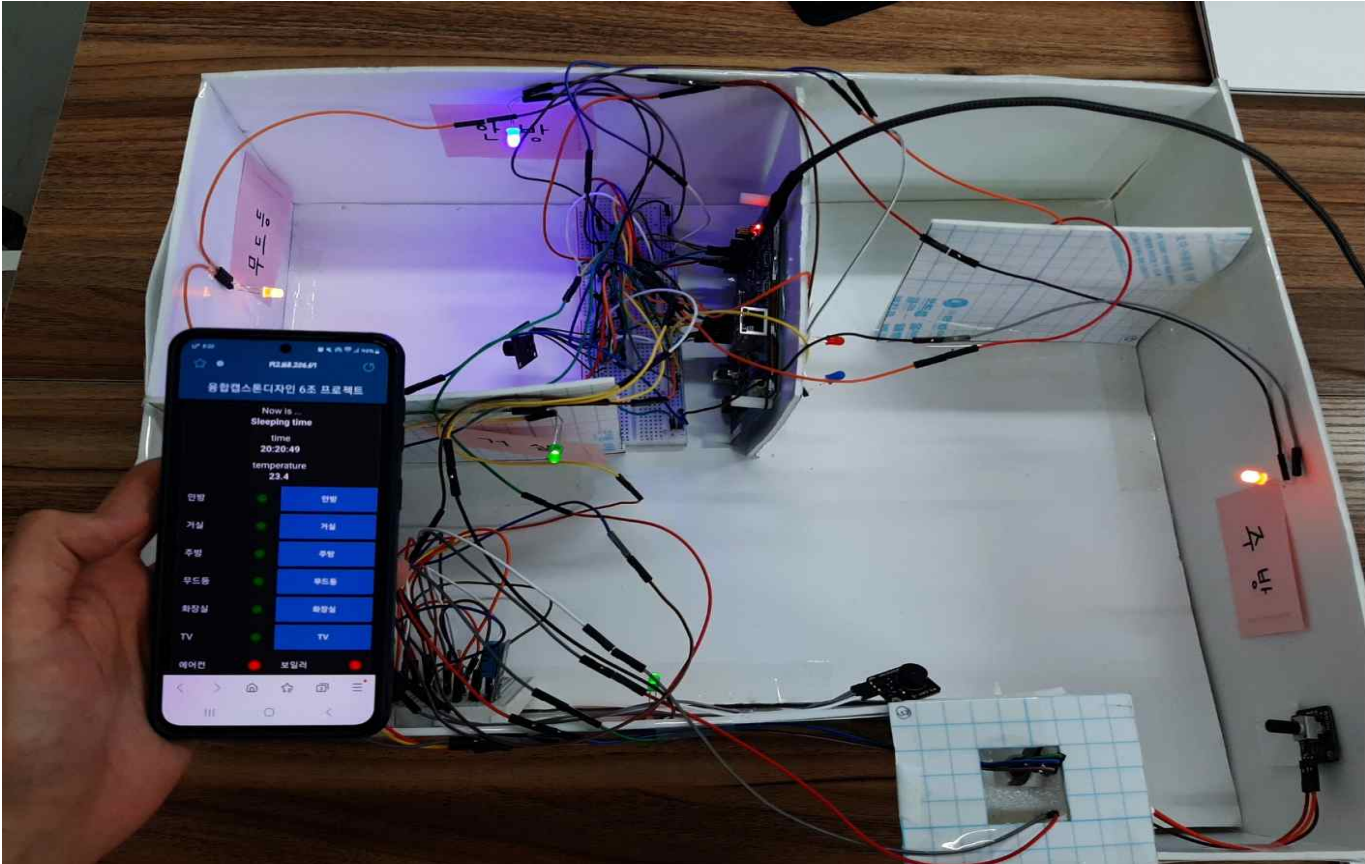
학 번: 201821094, 201820856

성 명: 안재혁, 이민혁

# 1. 개발 목표

- ARM Cortex-M7 아키텍처 기반인 stm32 보드를 이용하여, 현재 사람들에게 큰 이로움을 주고 있는 IoT 홈케어 서비스를 직접 구현해보는 것을 목표로 한다.
- 집에서 사용되는 센서와 유사한 기능을 가진 센서들을 설치하여 stm32 보드로 동작시키고, node-RED 를 사용하여 센서들을 직접 모바일에서 제어할 수 있는 dashboard를 제작한다.
- 집을 모형으로 직접 설계하면서 실제 집에서 설치 및 사용되는 센서들의 기능, 용도 등을 구체적으로 파악한다.

## ● 최종 개발 결과물

	
<p>하드웨어(집 모형)</p>	<p>모바일 dashboard</p>
	
<p>센서 및 LED 컨트롤</p>	

## 2. 팀 구성

201821094 안재혁	201820856 이민혁
<ul style="list-style-type: none"> <li>- 센서 스펙 파악 및 구매</li> <li>- 집 구조 및 실험 환경 설계</li> <li>- 보드 간 데이터 통신 연결</li> </ul>	<ul style="list-style-type: none"> <li>- 사용할 센서 파악</li> <li>- 센서 및 보드 연결도 설계</li> <li>- 각 센서에서 사용할 임계값 설정</li> <li>- IoT 클라우드, dashboard 제작</li> </ul>
<ul style="list-style-type: none"> <li>- 하드웨어(집) 모형 제작</li> <li>- 센서 연결 및 동작 코드 설계</li> <li>- 발표자료 제작, 관련 자료 분석</li> </ul>	

\* 위의 작성한 분담 업무들은 서로 도와가면서 유동적으로 진행하였습니다.

## 3. 개발 방법

### ① 기술 개발 방향

<최초 설정>

보안 기능	적외선 센서와 부저를 활용하여, 집주인이 취침 중일 때 현관으로 출입하는 사람을 감지하여 경고음을 준다.
전등 제어 기능	열 감지 센서(온도 센서)를 활용하여 침실에서 사람의 취침이 확인되면 집 안의 모든 전등 및 전자제품(TV)을 끄고, 무드등을 켜도록 한다.
가스 밸브 경고 기능	가스 밸브를 버튼 센서에 눌러게 함으로써 가스 밸브의 상태를 인식한다. 취침 시간 중 밸브가 일정 시간 이상 열려있으면 부저로 경고음을 준다.
자동 온도 조절 기능	적정 실내 온도 범위를 설정하고 설정 범위를 벗어나면 에어컨이나 보일러를 작동시킨다. 적정온도에 도달하면 자동으로 동작을 종료시킨다.
Node-RED를 활용한 모바일 조작	모든 센서의 데이터들은 Node-RED의 dashboard에서 확인이 가능하며, 버튼 인터페이스를 추가하여 사용자의 임의 조작이 가능하도록 한다.
시간에 따른 커튼 제어	회전 센서를 활용하여 커튼을 조작할 수 있게 하고, 취침 시간이 되면 커튼을 치고 기상시간이 되면 커튼을 걷도록 한다.

<변경 후 설정>

보안 기능	적외선 센서와 부저를 활용하여, 집주인이 취침 중일 때 현관으로 출입하는 사람을 감지하여 경고음을 준다.
전등 제어 기능	열 감지 센서(온도 센서)를 활용하여 침실에서 사람의 취침이 확인되면 집 안의 모든 전등 및 전자제품(TV)을 끄고, 무드등을 켜도록 한다.
가스 밸브 경고 기능	가스 밸브가 돌아간 각도에 따라 가스 밸브의 상태를 인식한다. 취침 시간 중 밸브가 일정 시간 이상 열려있으면 부저로 경고음을 준다.
자동 온도 조절 기능	적정 실내 온도 범위를 설정하고 설정 범위를 벗어나면 에어컨이나 보일러를 작동시킨다. 적정온도에 도달하면 자동으로 동작을 종료시킨다.
Node-RED를 활용한 모바일 조작	모든 센서의 데이터들은 Node-RED의 dashboard에서 확인이 가능하며, 버튼 인터페이스를 추가하여 사용자의 임의 조작이 가능하도록 한다.
시간에 따른 커튼 제어	—

\* 변경 항목은 기능 항목에서 적색 글씨로 나타내었습니다.

## ② 개발 기술 개요

<프로젝트에서 사용한 센서의 동작 원리>

### 1) PIR 센서

- 모델명 : 한글보드, PIR Motion Sensor
- 감지 거리 : 약 6m 미만
- 감지 각도 : 100도
- 출력 시간 : 감지시 3초
- 적외선으로 사람의 움직임을 감지한다.
- 움직임을 감지되면 1, 감지되지 않으면 0을 출력한다. 움직임을 감지하게 되면 기본적으로 현관 LED를 일정 시간 동안 작동시키게 되며, 만약 사용자가 자는 sleeping time에 센서가 감지되면 부저를 울리도록 설정하였다.

Pin Number	Name
1	OUT(Digital)
2	VCC
3	GND



### 2) 부저

- 모델명 : 한글보드, 부저 스피커 모듈
- 우리가 사용한 부저는 Piezo Passive Buzzer를 사용하며 TIM2의 PWM Generation CH1을 사용해서 부저를 작동시킨다. PWM 주파수는  $((\text{Internal Clock})/(\text{Prescaler} + 1)) / (\text{Period})$ 로 계산되고, DutyCycle(%)은  $(\text{Pulse})/(\text{Period}) \times 100$ 으로 계산되기 때문에 적절한 Prescaler와 Period를 설정하였다. TIM2는 APB1을 사용하며, 우리 보드 기준 APB1에 설정된 주파수는 42MHz였다. 따라서 Prescaler는 420-1로 설정하고, Period는 100으로 설정하였다. 그리고  $42\text{MHz} / (\text{PSC}(420) * \text{ARR}) = \text{음계별 표준 주파수}$ 를 의미하기 때문에,  $\text{ARR} = 100,000 / \text{음계별 주파수}$ 의 식을 사용하여 각 음계에 해당하는 ARR의 값을 설정한다. 레#, 파#의 소리를 사용하여 사이렌과 같은 소리를 부저에서 낼 수 있도록 하였다.

Pin Number	Name
1	IN
2	VCC
3	GND



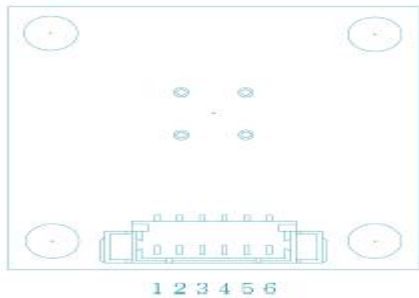


### 3) 온도 센서

- 모델명 : DTS-SIL300-B
- 대상 온도와 센서 온도를 동시에 측정
- SPI 디지털 통신으로 온도 데이터를 송수신
- Digital Resolution : 0.01도
- SPI Data Mode : Mode3(SCK rising edge data sampling, SCK idle High), Full duplex master
- SPI Clock : MAX 1MHz
- SPI bit order : MSB first
- Hardware NSS signal : Hardware NSS Output Signal
- 온도 센서를 사용하기 위해선 6개의 핀이 필요하며, Vcc(3.3V) 및 GND와 더불어 SPI 통신에 필요한 SCE, SCK, MOSI, MISO 핀이 필요하다.



<온도 센서 사진>



Pin Number	통신 방식
	SPI
1	GND
2	SCE
3	SCK
4	MOSI(SDI)
5	MISO(SDO)
6	3.3V

<Pin Assignment>

- 영상온도를 계산하기 위해 두 가지의 명령어 주소를 사용하였으며, 각각 대상 온도를 측정하는 0xA0와 센서 온도를 측정하는 0xA1을 사용하였다. 하나의 명령어 주소에는 상위 byte와 하위 byte를 저장하기 위한 두 개의 Byte buffer가 필요하며, 각 byte마다 SPI 통신을 통하여 값을 읽어들이고 두 값을 합쳐서 온도를 표현한다. 예시를 들면 아래 사진과 같다.

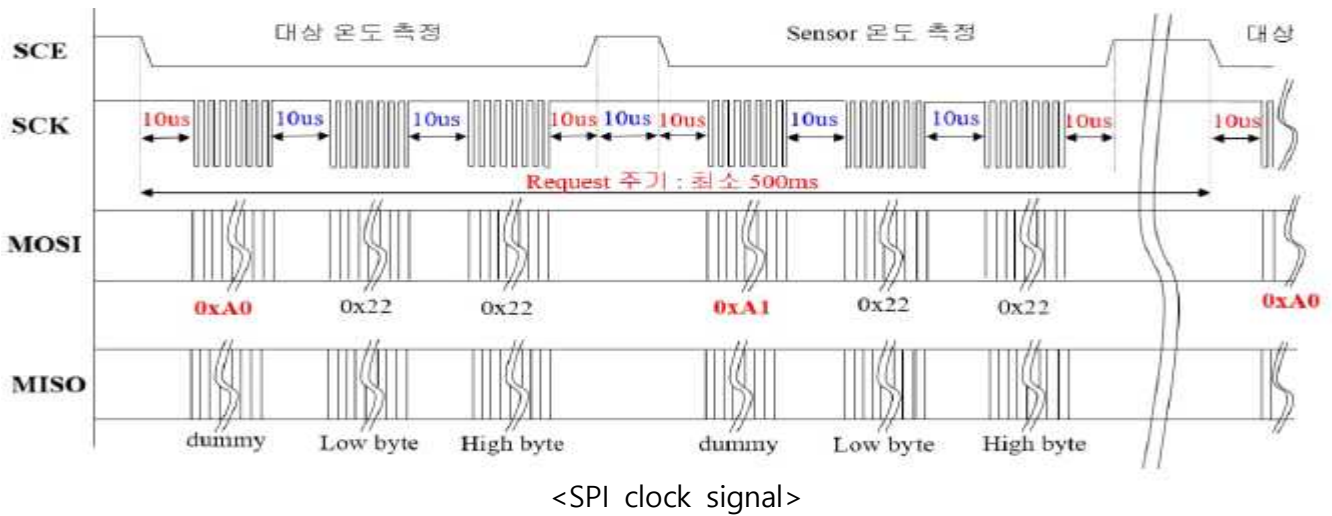
0xA0	0x42	0x0E
0xA1	0xC4	0x09

\* **object 온도 계산** : 상위 Byte(0x0E) + 하위 Byte(0x42) = 0x0E42 (Hex)  
=> 3650 (dec) 이며 이 값을 100으로 나누면 36.50 도입니다.

\* **sensor 온도 계산** : 상위 Byte(0x09) + 하위 Byte(0xC4) = 0x09C4 (Hex)  
=> 2500 (dec) 즉, 25.00 도입니다.

<온도값 통신 및 계산 방법>

- 먼저 온도 측정을 위한 명령어 주소는 0xA0와 0xA1을 순차적으로 보내야하며, 0xA0을 보내면 대상 온도를 측정하겠다는 signal을 준 것이다. 그 뒤로 10us의 간격을 두면서 상위 byte와 하위 byte의 값을 읽어들인다.(0xA1도 동일한 방법으로 통신한다.) SCK는 기본적으로 High 상태를 유지하며 Low로 변화하면서 값을 읽어들이기 시작한다. 이에 따라 CPOL은 1로 설정하며 SCK가 rising edge일 때 값을 읽어들이므로 CPHA는 2 Edge 상태로 설정해야 한다.



#### 4) 회전 센서

- 모델명 : 한글보드, 270도 회전센서 / 각도 감지
- 회전 각도 : 0~270도
- 내장 저항 : 20k옴
- 작동 전압 : DC 3.3V~5V
- 현재 센서의 돌아간 정도를 0~1023의 숫자로 나타낸다. 이 값을 3.78로 나누어 0~270°의 각도를 읽어 들일 수 있도록 scaling 한다. 이 각도가 10° 이상 돌아가게 되면 가스 밸브가 돌아간 것으로 판단하여 부저를 울리게 된다.

Pin Number	Name
1	OUT(Analog)
2	VCC
3	GND



- OUT 핀은 Analog에 대한 핀으로 설정하며 VCC는 3.3V로 설정한다.
- 센서의 기본적인 출력값은 0~1023이므로, resolution을 10 bit로 설정하면 오른쪽 사진과 같이 0~270도 사이의 값으로 출력되는 것을 알 수 있다.

```

Degree : 0
Degree : 69
Degree : 133
Degree : 171
Degree : 231
Degree : 270

```

<각도 측정값>

<프로젝트에서 사용한 기술>

1) USART 통신 – printf(), 두 보드 간의 통신

이번 프로젝트에서는 2개의 USART 통신 핀을 사용하였다. USART1과 USART6를 설정하였으며, 각각 printf() 출력값을 terminal로 확인하기 위한 용도와 보드 간의 데이터를 주고 받기 위한 용도로 사용하였다. 사용한 모드는 Asynchronous로 설정하였으며, USART1은 PA9(TX)와 PA10(RX)로 핀 설정하였고 USART6는 PC6(TX)와 PC7(RX)로 핀 설정하였다.

USART1 Mode and Configuration

Mode

Mode

Asynchronous

Hardware Flow Control (RS232)

Disable

☐ Hardware Flow Control (RS485)

Configuration

Reset Configuration

✔ DMA Settings

✔ User Constants

✔ Parameter Settings

✔ GPIO Settings

✔ NVIC Settings

Search Signals

Search (Ctrl+F)

Pin Name	Signal on Pin	GPIO output L...	GPIO mod
PA9	USART1_TX	n/a	Alternate Fu
PA10	USART1_RX	n/a	Alternate Fu

USART1

USART6 Mode and Configuration

Mode

Mode

Asynchronous

Hardware Flow Control (RS232)

Disable

☐ Hardware Flow Control (RS485)

Configuration

Reset Configuration

✔ DMA Settings

✔ User Constants

✔ Parameter Settings

✔ GPIO Settings

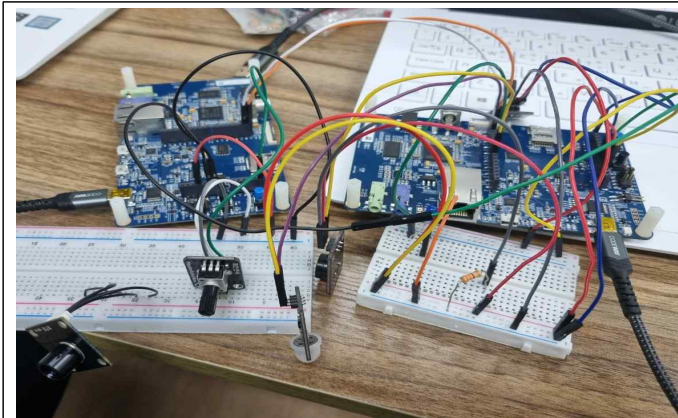
✔ NVIC Settings

Search (Ctrl+F)

Pin Name	Signal on Pin	GPIO output L...	GPIO mod
PC6	USART6_TX	n/a	Alternate Fu
PC7	USART6_RX	n/a	Alternate Fu

USART6

추가로 두 보드 간에 데이터를 송수신할 수 있도록 설정하였다. 각 보드의 D0와 D1 핀을 상호교차하여 연결해주었다.(TX, RX 핀이므로) Node-RED와 연결된 보드로부터 나머지 보드에게 현재 시각을 전송해주고, 시각을 전달받은 보드는 Sleeping time에 맞게 부저 및 적외선 센서를 동작하도록 설계하였다.



보드끼리 연결한 사진

COM8

18:39:36

18:39:36

18:39:37

18:39:37

18:39:38

18:39:38

18:39:39

18:39:40

18:39:40

18:39:41

18:39:41

현재 시각을 수신한 보드의 terminal

- 7 -

온도 센서를 동작시키기 위해 설정해야 하는 목록은 Mode, Hardware NSS Signal, Data Size, First Bit, Prescaler, CPOL, CPHA이다. 보드와 센서 간 양방향으로 통신해야 하므로 "Full-Duplex Master"모드로 설정하였고, 온도 센서가 값을 읽어들이며 보드로 전송하기 때문에 "Hardware NSS Output Signal"로 설정하였다. 이어서 buffer 당 1byte씩 받고 MSB부터 값을 읽어들이기 때문에, 8 bits의 데이터 크기와 MSB First로 설정하였다. 또한 센서의 데이터 최대 송수신 속도를 고려하여 prescaler를 64로 설정한 것과 CPOL 및 CPHA를 각각 High, 2 Edge로 설정하였다. 이 부분은 "<프로젝트에서 사용한 센서의 동작 원리> - 3) 온도 센서"에서 설명하였기에 자세한 내용은 생략하도록 하겠다.

Mode

Full-Duplex Master

Hardware NSS Signal

Hardware NSS Output ...

Configuration

Reset Configuration

DMA Settings

GPIO Settings

User Constants

NVIC Settings

Parameter Settings

Configure the below parameters :

Search (Ctrl+F)

Basic Parameters

Frame Format

Motorola

Data Size

8 Bits

First Bit

MSB First

Clock Parameters

Prescaler (for Bau...

64

Baud Rate

656.25 KBits/s

Clock Polarity (CP...

High

Clock Phase (CP...

2 Edge

Advanced Parameters

CRC Calculation

Disabled

NSS Signal Type

Output Hardware



### 3) ADC 컨버터 사용 – 회전 센서

회전 센서는 입력 부분의 회전 수치를 0~270 사이의 각도로 읽어들이는다. 하지만 처음 센서가 읽어들이는 값은 0~270 사이의 값이 아니라, 0~1023까지 이진수로 표현된 값이며 이를 원하는 수치로 해석하기 위해서는 resolution 기능을 설정해야한다. 본래 출력값은 1023(2의 10제곱)까지 나타나기 때문에 resolution은 10 bits로 설정해야 한다. 추가로 ADC를 위한 핀은 IN4를 사용하여 PF6를 설정하였다.

ADC3 Mode and Configuration

Mode

☐ IN3

☒ IN4

Configuration

Reset Configuration

☒ DMA Settings

☒ GPIO Settings

☒ User Constants

☒ NVIC Settings

☒ Parameter Settings

Configure the below parameters :

ADC\_Settings

Clock Prescaler

Resolution

Data Alignment

Scan Conversion ...

Continuous Conver..

Discontinuous Con

PCLK2 divided by 4

10 bits (13 ADC Clock cycl...

Right alignment

Disabled

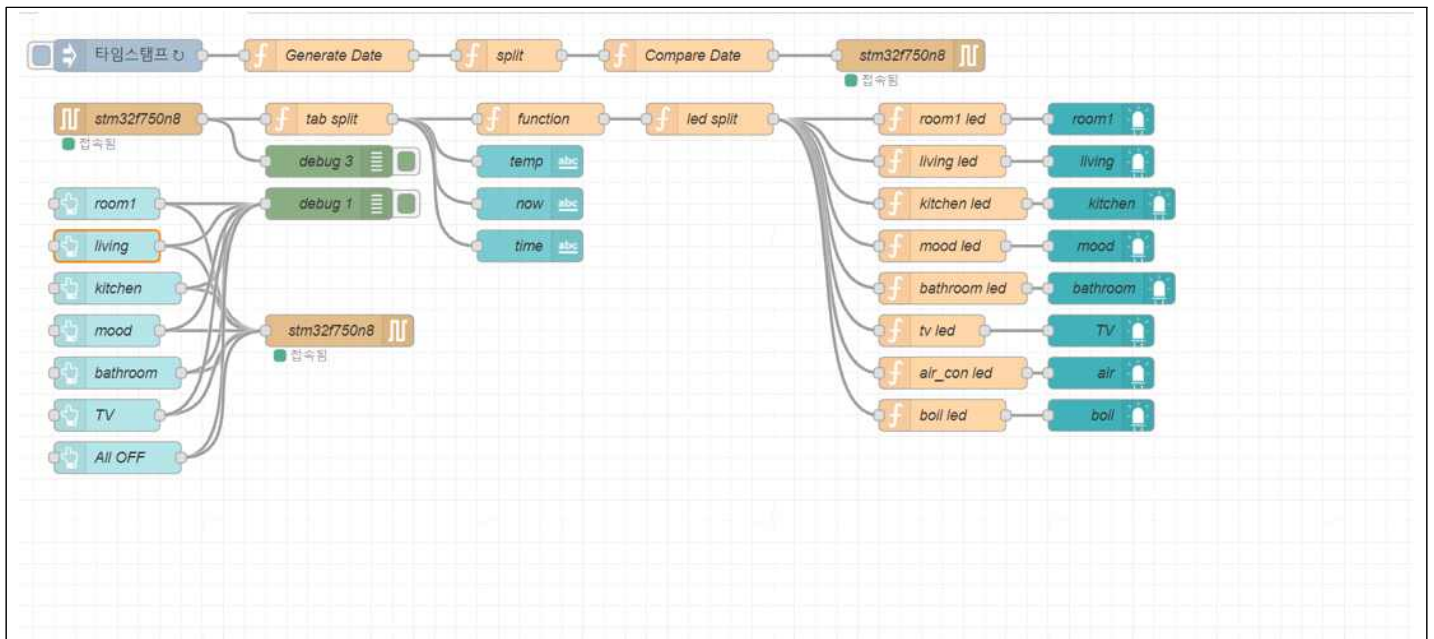
Disabled

Disabled

- 9 -

#### 4) Node-RED

먼저 하나의 stm32 보드와 Node-RED를 연결하여 센서값들을 dashboard에 나타내도록 하고, 반대로 dashboard에서 LED를 자유자재로 제어할 수 있도록 설계하였다. 전체 노드 구성도와 각 노드의 설정 내용 그리고 dashboard를 사진으로 아래와 같이 첨부하였다.



노드 구성도

**button의 노드 수정**

삭제 취소 완료

속성

Group: [융합캡스톤디자인 6조 프로젝트] 융합

Size: 3 x 1

Icon: optional icon

Label: 안방

Tooltip: optional tooltip

Color: optional text/icon color

Background: #4682B4

When clicked, send:

Payload: a000000

Topic: msg.topic

If msg arrives on input, emulate a button click: ☐

button node

**function의 노드 수정**

삭제 취소 완료

속성

이름: Compare Date

Setup On Start 코드 On Stop

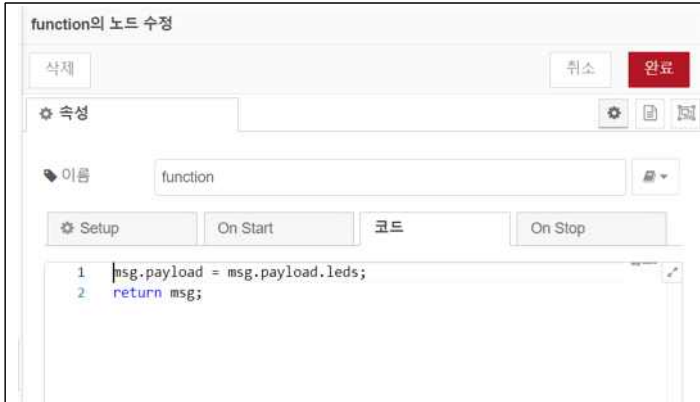
```

1 msg.payload = msg.payload.date;
2 return msg;

```

사용가능

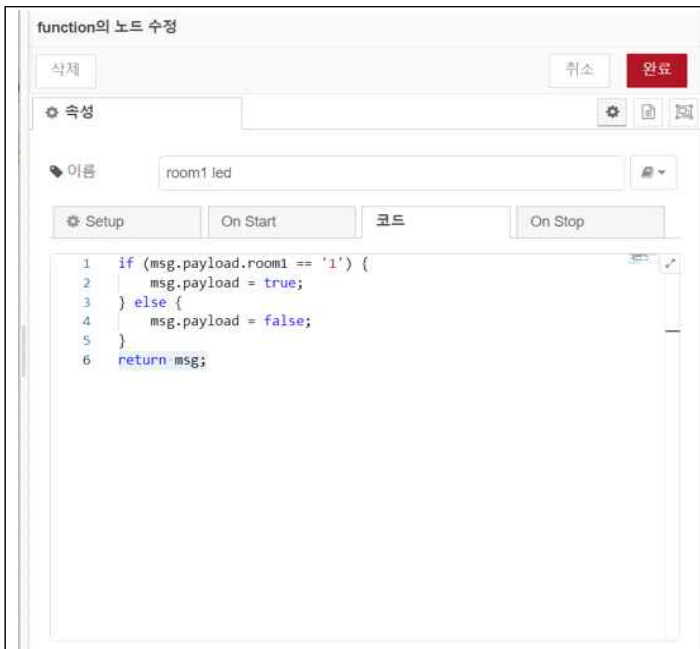
compare\_date node



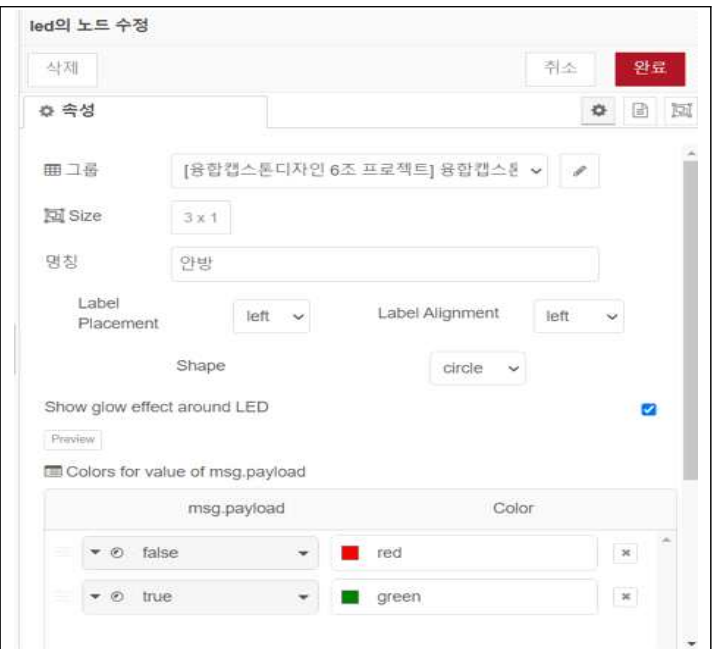
function node



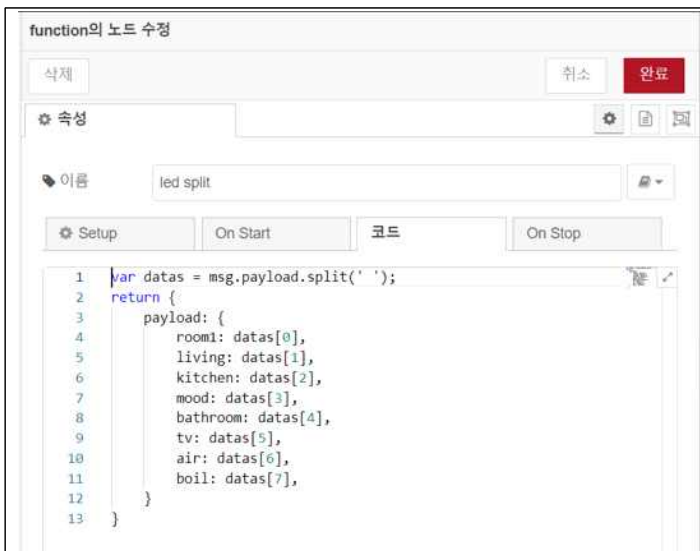
generate\_date node



led\_function node



led node



led\_split node



split node

text의 노드 수정

삭제 취소 완료

속성

Group [융합캡스톤디자인 6조 프로젝트] 융합?

Size auto

Label Now is ...

Value format {{msg.payload.text}}

Layout

label value label value label value

label value label value

Class Optional CSS class name(s) for widget

Name now

stat\_text node

text의 노드 수정

삭제 취소 완료

속성

Group [융합캡스톤디자인 6조 프로젝트] 융합?

Size auto

Label temperature

Value format {{msg.payload.temp}}

Layout

label value label value label value

label value label value

Class Optional CSS class name(s) for widget

Name temp

temp\_text node

text의 노드 수정

삭제 취소 완료

속성

Group [융합캡스톤디자인 6조 프로젝트] 융합?

Size auto

Label time

Value format {{msg.payload.time}}

Layout

label value label value label value

label value label value

Class Optional CSS class name(s) for widget

Name time

time\_text node

inject의 노드 수정

삭제 취소 완료

속성

이름 이름

msg. payload = timestamp

msg. topic = 0\_2

+ 추가 inject now

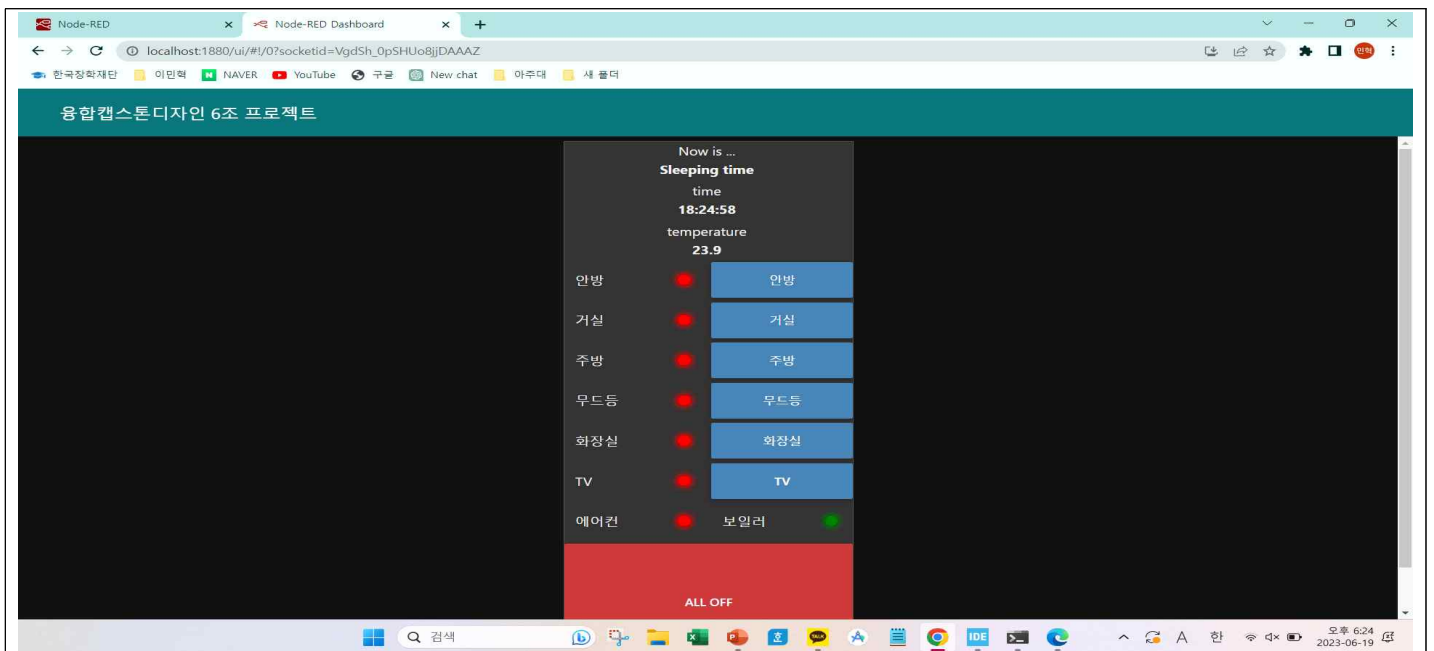
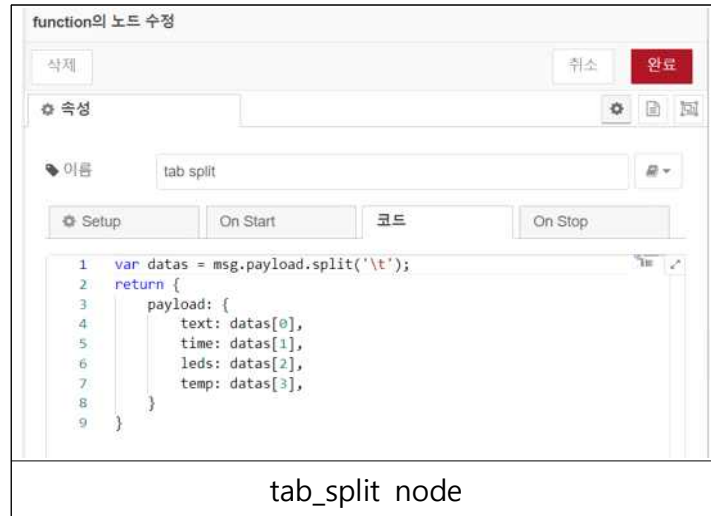
☒ Node-RED시작의 0.1 초 후, 아래를 실행

반복 지정한 시간간격

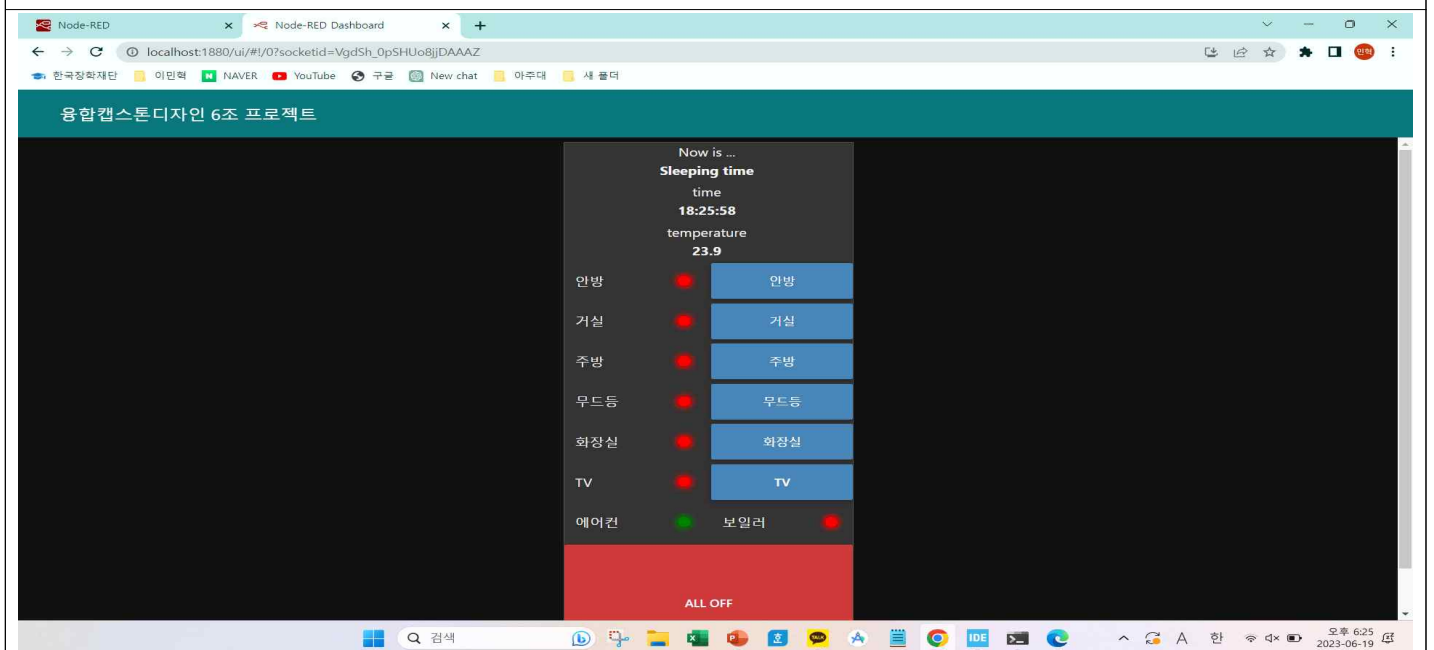
시간각격 1 초

timestamp node





보일러 임계값 25도로 변경 후의 dashboard



에어컨 임계값 23도로 변경 후의 dashboard

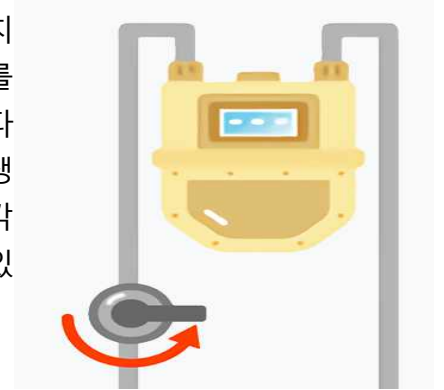
### ③ 독창성 및 도전성

#### <독창성>

IoT 홈케어 서비스는 실제로도 구현된 사례들이 있지만, 이번 프로젝트에서는 기존의 구현된 사례에서 벗어나 사례에 없는 기능을 구현하고 창의적인 방식으로 센서를 사용하는 것에 독창성을 두었다.

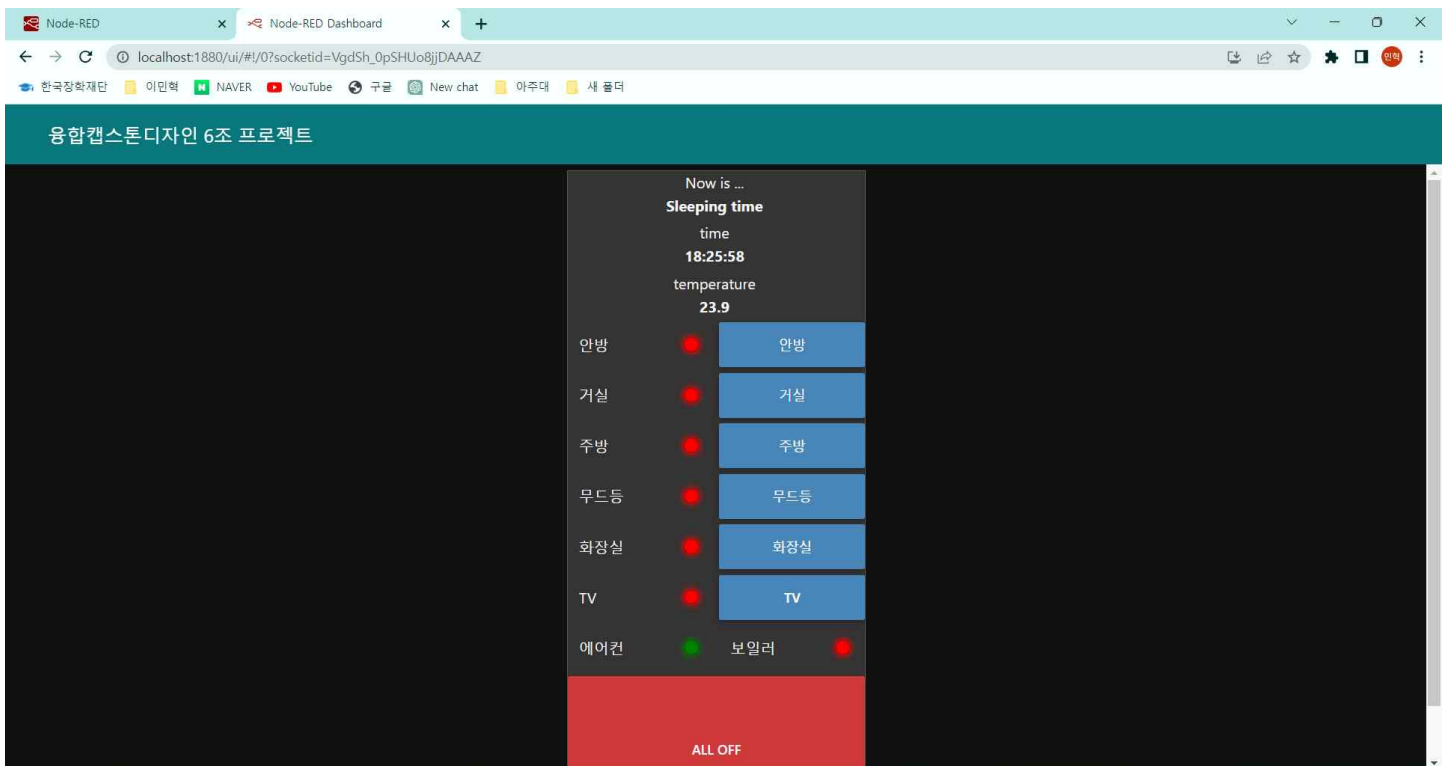
#### 1) 가스 밸브를 회전 센서로 구현

본래 가스 밸브는 터치 센서로 이용하여, 밸브가 닫혔을 경우 밑에 위치한 터치 센서를 누르도록 설계 방향을 잡았었다. 하지만 이를 회전 센서를 대체함으로써, 가스 밸브의 움직임과 회전 센서의 움직임이 매우 유사하다고 생각하여 각도에 따라 가스 밸브의 open/close를 구현하기 편하다고 생각하였다. 따라서 회전 센서가 읽어들이는 각도는 곧 가스 밸브가 열린 각도로 판단하여, 이에 맞게 컨트롤 하도록 설계한 점에 있어서 독창성이 있다고 할 수 있다.



#### 2) 현재 시각을 Node-RED로 불러와 stm32와 동기화

이번 프로젝트에서 구현한 dashboard는 아래 사진과 같이 수면 시간, 현재 시각, 집안 온도, 각 LED의 on/off 상태, 에어컨과 보일러 on/off 상태를 확인할 수 있도록 구성하였다. 또한 LED는 dashboard에서도 직접 구현할 수 있도록 설계하였고 ALL OFF 버튼을 만들어 집 내의 모든 LED를 off 할 수 있도록 제작하였다. dashboard에 어떤 항목을 display할지 정하고 기능을 구현하는 데 있어서, 6조의 독창성이 들어가있다고 할 수 있다. 특히나 sleeping time의 유무는 다른 사례에서도 절대 찾아볼 수 없는 항목인 만큼 독창적인 항목이라 생각한다.



<dashboard 구성도>

### 3) 온도 센서의 사용

이번 프로젝트에서 사용한 온도 센서는 대상 온도와 주변 온도를 측정할 수 있는 복합적 기능이 있다. 여기서 대상 온도를 측정하는 기능은 사람이 침대에 누워있는 것을 인식하는 상황에서 사용할 수 있도록 설계하였다. 침대 위에서 사람을 인식하기 위해서는 온도 센서의 기능을 이용할 수밖에 없다고 판단하였다. 이에 따라 온도 센서가 사람의 체온을 감지하면, 침대에 사람이 누워있는 것으로 인식하고 사용자가 설정한 시간인 sleeping time에 해당하면 집안의 모든 전등과 전자제품을 turn off하고 무드등을 켜는 것으로 구현하였다. 이를 구현한 사례는 어디에도 없으며 복합적 기능의 온도 센서를 적절히 이용하고 이에 따른 turn off/on 기능을 연결지어 구현한 점에 대하여 독창성이 있다고 할 수 있다. 오른쪽 사진은 센서에 손을 가까이함으로써, 온도 센서가 체온을 인식했을 경우 무드등이 켜지는 사진이다.



### <도전성>

비록 다른 프로젝트에 비해 사전에 구현된 사례가 어느 정도 있는 편인 주제이다. 하지만 stm32 보드를 통해 IoT 홈케어 서비스를 직접 우리 손으로 구현해보고, 데이터를 송수신하기 위한 통신 방식과 센서의 동작 원리 등을 확실하게 인지하여 위와 같은 창의적인 방식으로 실현해내는 것에 대하여 도전성이 있다고 할 수 있다.

## 4. 개발 내용

### ① 개발 내용

<소스코드 분석>

우선 소스코드 분석을 작성하기 전, 이번 프로젝트에서 사용한 두 개의 보드에 대하여 각 보드 별로 사용한 센서로 나누어 작성하겠습니다.

보드 1의 기능 : 부저, PIR 센서, 회전 센서 작동

보드 2의 기능 : 온도 센서 작동, 보드 1로의 데이터 전송, Node-RED로의 데이터 전송

- 보드 1

1) 부저

```
/* USER CODE BEGIN PV */
enum notes { // 부저 음계 주파수에 해당하는 ARR 값 설정
    //C = 956,
    D = 170, // 레#
    //E = 758,
    F = 135, // 파#
    //G = 638
};
/* USER CODE END PV */

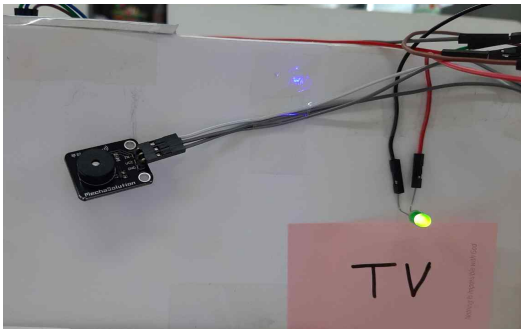
/* USER CODE BEGIN 0 */
uint16_t bell[] = { F, D, F, D, F, D, F, D, F, D, F, D, F, D, F, D, F, D, F, D, F, D,
    F, D, F, D, F, D, F, D, F, D, F, D, F, D, F, D, F, D }; // 사이렌 악보
uint8_t bell_length = sizeof(bell) / sizeof(uint16_t); // for문에 사용하도록 구한 length
void buzzer_on(void) // 부저를 작동시키는 함수
{
    printf("buzzer on! \n\n"); // 부저가 on되면 print
    HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1); // 타이머 스타트
    for (int i = 0; i < bell_length; i++) {
        TIM2->ARR = bell[i]; // 악보에 따라 ARR 값 변경
        TIM2->CCR1 = TIM2->ARR / 2; // 듀티사이클 50%로 고정
        HAL_Delay(500); // Inter-tone delay
        TIM2->CCR1 = 0;
        HAL_Delay(10);
        TIM2->CCR1 = TIM2->ARR / 2;
    }
    HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1); // 타이머 스톱
}
```



(종락)

```
while (1)
{
    // 회전 센서의 각도가 10도 이상 돌아가거나 sleeping time일때 PIR 센서가 반응하면 부저 울림
    if (vout >= 10) buzzer_on(); // 각도가 10도 이상이면 부저 울림

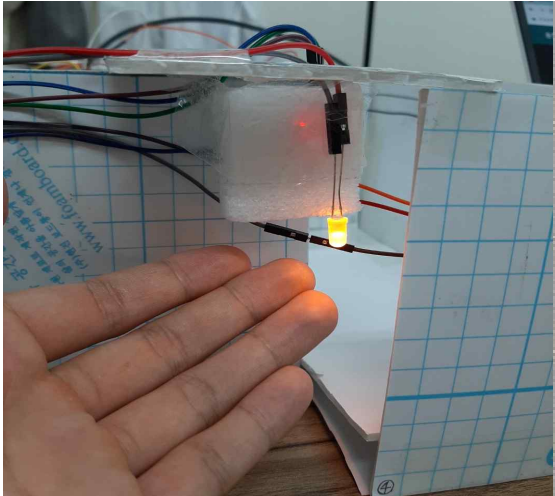
    if (pir == GPIO_PIN_SET) // PIR 센서의 값이 1이면
    {
        HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET); // 현관 LED ON
        if ((secs >= 0 && secs < 86400) || (secs >= 0 && secs < 21600)) // 지금이 sleeping time이라면
        {
            printf("Sleeping time WnWr");
            buzzer_on(); // 부저 ON
        }
    }
}
```



## 2) PIR 센서

```
/* USER CODE BEGIN 0 */
static int pir = 0;
...(전략)
while (1)
{
    pir = HAL_GPIO_ReadPin(PIR_GPIO_Port, PIR_Pin); // PIR 센서의 값 읽음
    if (pir == GPIO_PIN_SET) // PIR 센서의 값이 1이면
    {
        HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET); // 현관 LED ON

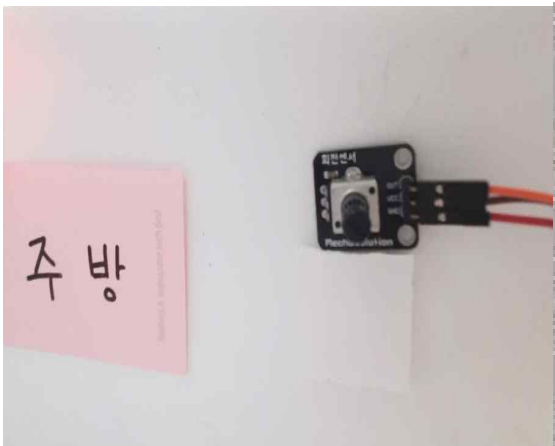
        if ((secs >= 0 && secs < 86400) || (secs >= 0 && secs < 21600)) // 지금이 sleeping time이라면
        {
            printf("Sleeping time WnWr");
            buzzer_on(); // 부저 ON
        }
        HAL_Delay(1000);
    }
    else HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET); // PIR 센서의 값이 0이면 LED OFF
}
...(후락)
```



```
PIR : 0
PIR : 0
PIR : 0
PIR : 0
PIR : 0
PIR : 0
PIR : 1
Sleeping time
```

### 3) 회전 센서

```
HAL_ADC_Start(&hadc3); // ADC converter Ready
HAL_ADC_PollForConversion(&hadc3, HAL_MAX_DELAY); // ADC를 사용하기 위해 대기
value = HAL_ADC_GetValue(&hadc3); // ADC로부터 얻은 값을 value에 저장
HAL_Delay(100); // 100ms delay
vout = (int)(value / 3.78); // ADC로부터 얻은 값을 3.78로 나누어 vout에 저장
```



```
Degree : 0
Degree : 69
Degree : 133
Degree : 171
Degree : 231
Degree : 270
```

### 4) 보드 2로부터 데이터 수신 후 시간 변환

```
HAL_UART_Receive_IT(&huart6, rx_buffer, sizeof(rx_buffer)); // 보드 2로부터 현재 시각 수신

int i;
// 현재 시각을 초 단위로 변환
for (i = 0; i < 7; i += 3)
{
    time[i / 3] = (rx_buffer[i] - 48) * 10 + (rx_buffer[i + 1] - 48);
}
secs = time[0] * 3600 + time[1] * 60 + time[2];
```

- 보드 2

## 1) 온도 센서

```
int _write(int fd, char* ptr, int len) // 시리얼 모니터에 출력하기 위한 함수
{
    HAL_UART_Transmit(&huart1, (unsigned char*)ptr, len, HAL_MAX_DELAY);
    return len;
}

HAL_StatusTypeDef SPI_TransmitReceive(SPI_HandleTypeDef* hspi, uint8_t* txData, uint8_t* rxData, uint16_t size,
uint32_t timeout)
{
    HAL_StatusTypeDef status;

    status = HAL_SPI_Transmit(hspi, txData, size, timeout);
    if (status != HAL_OK)
        return status;

    return HAL_SPI_Receive(hspi, rxData, size, timeout);
}
...(중략)
#define SPEED_1MHz 1000000
#define SPI_MODE3 SPI_MODE_CPOL1_CPHA1

#define EN_HIGH HAL_GPIO_WritePin(EN_GPIO_Port, EN_Pin, GPIO_PIN_SET)
#define EN_LOW HAL_GPIO_WritePin(EN_GPIO_Port, EN_Pin, GPIO_PIN_RESET)

#define OBJECT 0xA0 // 대상 온도 측정 명령어 주소
#define SENSOR 0xA1 // 센서 온도 측정 명령어 주소

volatile uint32_t* LAR = (uint32_t*)0xE0001FB0; // printf()를 위한 레지스터 추가 설정

static int16_t iSensor, iObject; // 대상 온도와 센서 온도 데이터를 저장하기 위한 변수
/* USER CODE END PD */
(중략)

uint32_t DWT_Delay_Init(void)
{
    /* Disable TRC */
    CoreDebug->DEMCR &= ~CoreDebug_DEMCR_TRCENA_Msk; // ~0x01000000;
    /* Enable TRC */
    CoreDebug->DEMCR |= CoreDebug_DEMCR_TRCENA_Msk; // 0x01000000;
    *LAR = 0xC5ACCE55;
    /* Disable clock cycle counter */
    DWT->CTRL &= ~DWT_CTRL_CYCCNTENA_Msk; //~0x00000001;
    /* Enable clock cycle counter */
    DWT->CTRL |= DWT_CTRL_CYCCNTENA_Msk; //0x00000001;

    /* Reset the clock cycle counter value */
    DWT->CYCCNT = 0;
```

```

/* 3 NO OPERATION instructions */
__ASM volatile ("NOP");
__ASM volatile ("NOP");
__ASM volatile ("NOP");

SysTick_Config(0);

/* Check if clock cycle counter has started */
if (DWT->CYCCNT)
{
    return 0; /*clock cycle counter started*/
}
else
{
    return 1; /*clock cycle counter not started*/
}
}

__STATIC_INLINE void delay(volatile uint32_t microseconds)
{
    uint32_t clk_cycle_start = DWT->CYCCNT;

    /* Go to number of cycles for system */
    microseconds *= (HAL_RCC_GetHCLKFreq() / 1000000);

    /* Delay till end */
    while ((DWT->CYCCNT - clk_cycle_start) < microseconds);
}

int16_t SPI_COMMAND(uint8_t cCMD) // 온도 측정하는 함수
{
    uint8_t T_high_byte, T_low_byte;
    uint8_t Data_buf[2];
    Data_buf[0] = 0x22;
    Data_buf[1] = 0x22;

    HAL_GPIO_WritePin(nss_GPIO_Port, nss_Pin, GPIO_PIN_RESET);
    delay(10); // Delay 1ms
    HAL_SPI_Transmit(&hspi2, &cCMD, 1, HAL_MAX_DELAY); // Transfer 1st Byte
    delay(10); // Delay 1ms
    HAL_SPI_TransmitReceive(&hspi2, Data_buf, &T_low_byte, 1, HAL_MAX_DELAY);
    delay(10);
    HAL_SPI_TransmitReceive(&hspi2, Data_buf + 1, &T_high_byte, 1, HAL_MAX_DELAY);
    HAL_Delay(1); // Delay 1ms
    HAL_GPIO_WritePin(nss_GPIO_Port, nss_Pin, GPIO_PIN_SET);

    return (T_high_byte << 8 | T_low_byte); // Return temperature value
}

/* USER CODE END 0 */

```



(중략)

```
main(void)
```

```
/* USER CODE BEGIN 2 */
```

```
double in_temp = 0.0; // 온도 저장 변수 초기화
```

```
double ob_temp = 0.0;
```

```
/* USER CODE END 2 */
```

```
/* USER CODE BEGIN WHILE */
```

```
DWT_Delay_Init(); // 딜레이 함수 초기화
```

```
while (1)
```

```
{
```

```
    iSensor = SPI_COMMAND(SENSOR); // 주변 온도(센서 온도) 측정
```

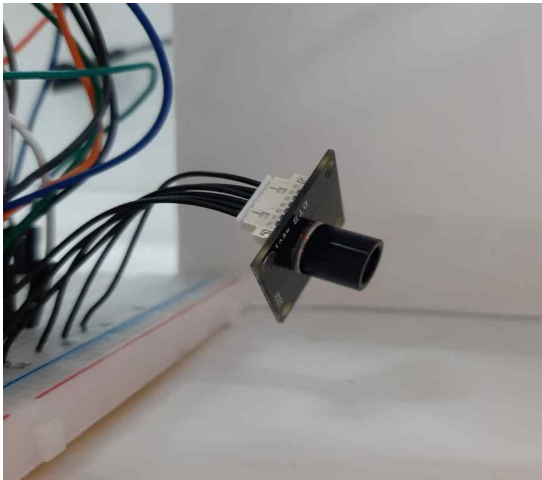
```
    delay(30);
```

```
    iObject = SPI_COMMAND(OBJECT); // 타겟 온도 측정
```

```
    in_temp = (double)iSensor / 100;
```

```
    ob_temp = (double)iObject / 100;
```

```
(후략)
```



```
sensor : 25.9  
object : 34.9  
Sensor : 25.9 , Object : 34.9  
sensor : 25.9  
object : 34.9  
Sensor : 25.9 , Object : 34.9  
sensor : 26.0  
object : 27.7  
Sensor : 26.0 , Object : 27.7  
sensor : 26.0  
object : 27.4  
Sensor : 26.0 , Object : 27.4
```

## 2) Node-RED 및 하드웨어 연결 동작

\* Node-RED로부터 데이터를 수신받는 코드

```
#define MAX_RX_BUFFER_SIZE 8
unsigned char rxData[9] = { 0 }; // Node-RED에서 받아온 정보를 저장하는 문자열
unsigned char Data[9] = { 0 }; // 시간 정보를 저장해서 다른 보드로 보내기 위한 문자열
unsigned int time[3]; // 시간 정보를 시, 분, 초로 나누기 위한 배열
unsigned int secs = 0; // 현재 시간을 초로 변환해서 저장하는 변수
unsigned int number = 0; // 문자(아스키코드)를 숫자로 변환한 결과를 저장하는 변수(case문 적용하기 위함)
int room1 = 0; // 각 방의 LED 상태를 저장하는 변수
int living = 0;
int kitchen = 0;
int mood = 0;
int bathroom = 0;
int tv = 0;
int air_con = 0;
int boil = 0;

void HAL_UART_RxCpltCallback(UART_HandleTypeDef* huart) {
    if (huart == &huart1) { // Node-RED와 통신하면
        if (!isdigit(rxData[0])) { // 첫 번째 데이터가 문자인지 숫자인지 검사
            number = rxData[0] - 97; // 문자(a~g)를 숫자로 변환
            memset(rxData, 0, sizeof(rxData));
            switch (number) // case에 따라 눌린 버튼에 해당하는 동작 실행
            case 0:
                //room1
                if (room1 == 1) { // 현재 상태에 대해 toggle
                    HAL_GPIO_WritePin(room1_GPIO_Port, room1_Pin, GPIO_PIN_RESET);
                    room1 = 0;
                }
                else if (room1 == 0) {
                    HAL_GPIO_WritePin(room1_GPIO_Port, room1_Pin, GPIO_PIN_SET);
                    room1 = 1;
                }
                break;
            case 1:
                //living
                if (living == 1) {
                    HAL_GPIO_WritePin(living_GPIO_Port, living_Pin, GPIO_PIN_RESET);
                    living = 0;
                }
                else if (living == 0) {
                    HAL_GPIO_WritePin(living_GPIO_Port, living_Pin, GPIO_PIN_SET);
                    living = 1;
                }
                break;
            ... (중략) : case 2 ~ case 5까지 실행 방식 동일(kitchen, mood, bathroom, tv)
```

```

default: // ALL OFF 버튼
    HAL_GPIO_WritePin(room1_GPIO_Port, room1_Pin, GPIO_PIN_RESET); // room1 living kitchen
bathroom TV
    room1 = 0;
    HAL_GPIO_WritePin(living_GPIO_Port, living_Pin, GPIO_PIN_RESET);
    living = 0;
    HAL_GPIO_WritePin(kitchen_GPIO_Port, kitchen_Pin, GPIO_PIN_RESET);
    kitchen = 0;
    HAL_GPIO_WritePin(bathroom_GPIO_Port, bathroom_Pin, GPIO_PIN_RESET);
    bathroom = 0;
    HAL_GPIO_WritePin(TV_GPIO_Port, TV_Pin, GPIO_PIN_RESET);
    tv = 0;
    HAL_GPIO_WritePin(mood_GPIO_Port, mood_Pin, GPIO_PIN_RESET); // mood
    mood = 0;
}
}
else if ((rxData[0] == 48) || (rxData[0] == 49) || (rxData[0] == 50)) { // 첫번째가 0또는1또는 2라면(시간이면)
    for (int i = 0; i < 9; i++) Data[i] = rxData[i]; // 다른 보드에 전송할 시간 데이터 저장
    for (int i = 0; i < 7; i += 3) time[i / 3] = (rxData[i] - 48) * 10 + (rxData[i + 1] - 48);
    secs = time[0] * 3600 + time[1] * 60 + time[2]; // 현재 시각을 초로 변환
}
HAL_UART_Transmit_IT(&huart6, rxData, sizeof(Data)); // 다른 보드에 시간 정보 보냄
}
}
...
(중략)
...
while (1)
{
    HAL_UART_Receive_IT(&huart1, rxData, MAX_RX_BUFFER_SIZE); // Node-RED에서 데이터 받음(시간 정보)
    (후략)
}

```

\* 현재 상태를 보고 LED 제어 및 데이터 시리얼모니터(Node-RED)로 보냄

```
unsigned int sleep_state = 0; // LED OFF를 한 번만 적용하기 위한 상태변수
...
(종락)
...
while (1)
{
    if (sleep_state == 0) // for the first time
    {
        if (ob_temp >= 32) // temperature check
        {
            if ((secs >= 59400 && secs < 86400) || (secs >= 0 && secs < 21600)) // sleeping time
            {
                HAL_GPIO_WritePin(room1_GPIO_Port, room1_Pin, GPIO_PIN_RESET); // room1 living kitchen
bathroom TV
                room1 = 0;
                HAL_GPIO_WritePin(living_GPIO_Port, living_Pin, GPIO_PIN_RESET);
                living = 0;
                HAL_GPIO_WritePin(kitchen_GPIO_Port, kitchen_Pin, GPIO_PIN_RESET);
                kitchen = 0;
                HAL_GPIO_WritePin(bathroom_GPIO_Port, bathroom_Pin, GPIO_PIN_RESET);
                bathroom = 0;
                HAL_GPIO_WritePin(TV_GPIO_Port, TV_Pin, GPIO_PIN_RESET);
                tv = 0;
                HAL_GPIO_WritePin(mood_GPIO_Port, mood_Pin, GPIO_PIN_SET); // mood
                mood = 1;
                sleep_state++;
            }
        }
    }
    if (in_temp <= 15) {
        HAL_GPIO_WritePin(boil_GPIO_Port, boil_Pin, GPIO_PIN_SET); // 보일러 작동
        boil = 1;
    }
    else if (in_temp >= 27) {
        HAL_GPIO_WritePin(air_con_GPIO_Port, air_con_Pin, GPIO_PIN_SET); // 에어컨 작동
        air_con = 1;
    }
    else {
        HAL_GPIO_WritePin(air_con_GPIO_Port, air_con_Pin, GPIO_PIN_RESET); //OFF
        air_con = 0;
        HAL_GPIO_WritePin(boil_GPIO_Port, boil_Pin, GPIO_PIN_RESET);
        boil = 0;
    }
    if ((secs >= 59400 && secs < 86400) || (secs >= 0 && secs < 21600)) // in sleeping time
    {
        printf("Sleeping time"); // 현재 상태 print
        printf("\n");
    }
}
```

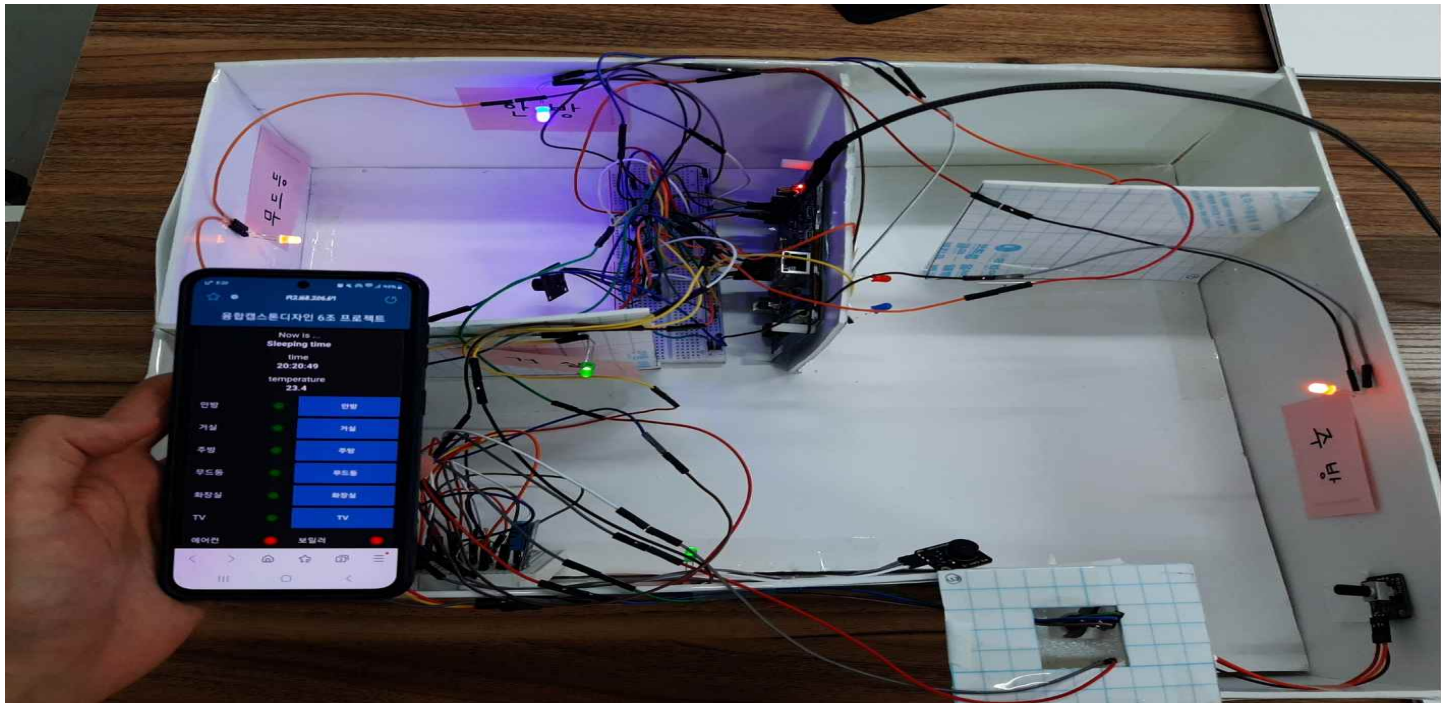


```

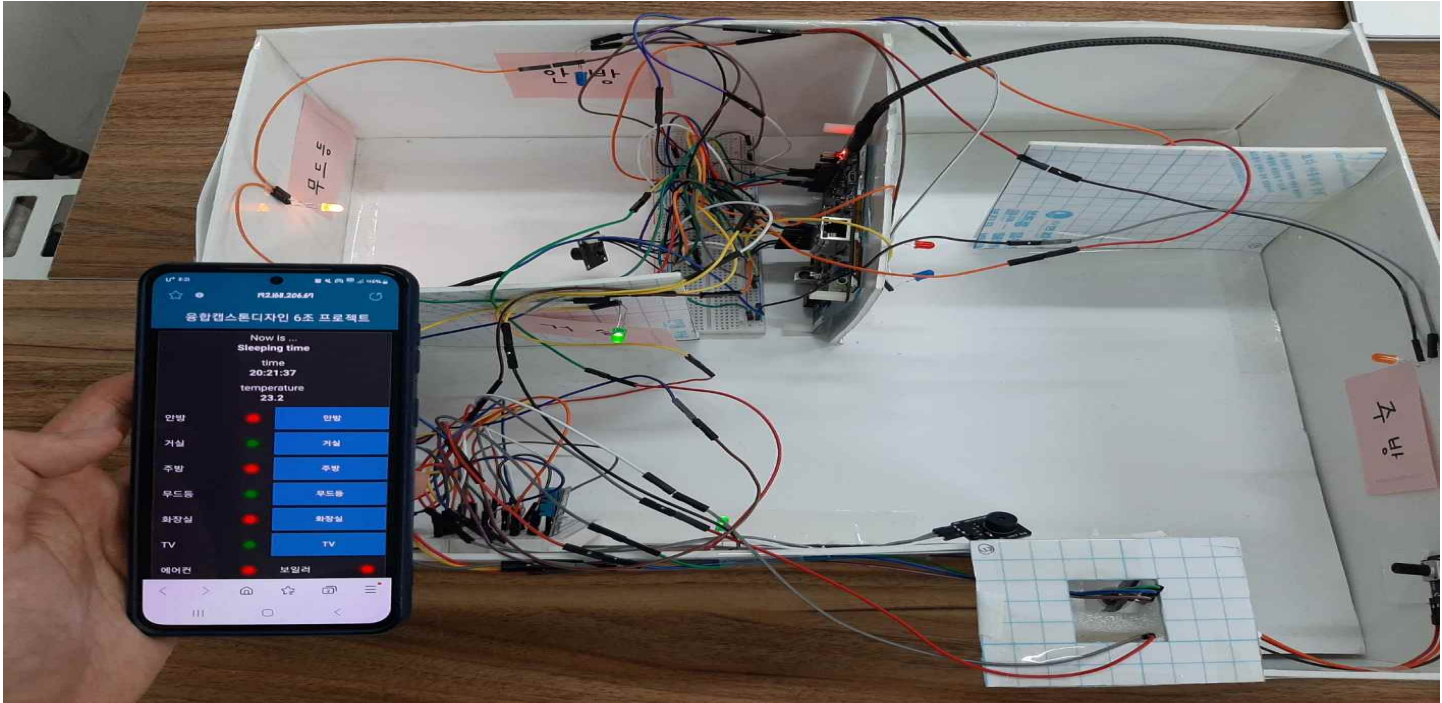
else
{
    printf("Not sleeping time");
    printf("Wt");
}
printf("%s", Data); // print time
printf("Wt");
printf("%d %d %d %d %d %d %d %d", room1, living, kitchen, mood, bathroom, tv, air_con, boil); // states of
led
printf("Wt");
printf("%5.1f", in_temp); // print temperature
printf("Wn"); // 구분자
HAL_Delay(500);

(후략)

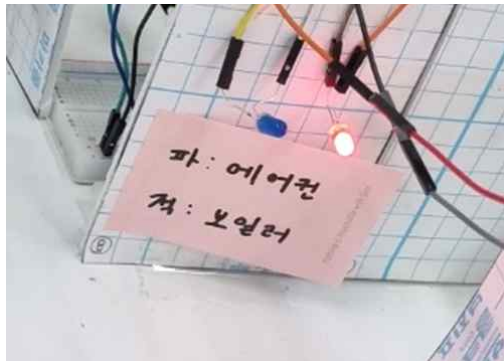
```



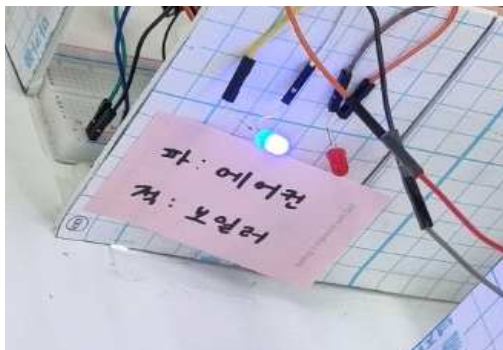
<모든 구역의 LED가 켜졌을 경우>



<LED를 절반 구역만 켜올 경우>



보일러 켜졌을 경우



에어컨 켜졌을 경우

## <개발 중 발생한 문제점 및 해결 방안>

### 1) 온도 센서 SPI 통신

프로젝트에서 사용한 온도 센서는 SPI 통신을 기반으로 데이터를 주고받는다. 센서의 datasheet를 기반으로 SPI 세부사항을 알맞게 설정하고 아두이노와 라즈베리파이의 코드를 기반으로 stm32 코드를 새롭게 작성하였으나, 온도가 0.0도로 출력되는 문제점이 발생하였다. 해당 문제를 해결하기 위해 총 2주의 시간을 소요하였으며, 이를 해결하기 위해 시도한 방안은 아래와 같다.

#### - Prescaler 설정

해당 센서에서의 최대 통신 속도는 1MHz이다. 따라서 Prescaler를 64로 설정하여 데이터 통신 속도를 최대 속도보다 아래인 656.24 KBits/s로 설정하여야 했다.

#### - Hardware NSS signal 설정

사용하고 있는 보드에서 SPI 통신을 사용하고 있는 센서는 온도 센서 하나뿐이었고, 측정한 온도 데이터를 읽어들이어 보드로 전송하기 때문에 Output signal로 설정해야 했다.

#### - HAL\_SPI\_TransmitReceive(), HAL\_SPI\_Transmit(), HAL\_SPI\_Receive() 함수 사용

처음 대상 온도 및 센서 온도 값을 읽어들이기 위해 0xA0와 0xA1을 보내야했고 그 뒤 두 개의 byte buffer에 대한 데이터를 담아야 한다. 따라서 데이터를 송수신하는 함수를 구현해야 했는데, 처음에는 Transmit()과 Receive()를 구현하기도 하였고 TransmitReceive()를 구현하기도 하였다. 하지만 데이터를 전송하는 것은 잘 되었으나 보드로 데이터를 받는 과정이 잘 안되었으며 Handler 코드로 빠지기도 하였다. 이를 해결하기 위해 delay와 상, 하위 byte를 받도록 코드를 별도로 설계하였다.

#### - us 단위 delay() 함수 구현

온도 센서의 데이터시트를 살펴보면, 각 버퍼를 송수신할 때마다 10us의 딜레이가 있어야한다. 또한 대상 온도와 센서 온도 사이에서 30us의 딜레이가 있어야 한다. 이에 따라 HAL\_Delay() 함수는 ms 단위로 구현되어 있었기 때문에, us 단위로 delay를 해주는 함수가 필요했다. 그리하여 그에 대한 함수를 따로 구현하였으며, 초반에는 잘 동작하지 않았으나 LAR 레지스터를 추가해준 이후로 디버깅을 하였을 때 올바르게 작동하였다.

### 2) Stm32cubemonitor

Stm32cubemonitor는 Node-RED와 매우 유사한 프레임워크를 가지고 있다. 따라서 Node-RED에 stm32 보드가 호환되도록 만들어진 프로그램이라 생각하여 이를 적극적으로 이용하려 하였다. stm32cubemonitor로 stm32 보드를 연결하여 dashboard로 나타내는 과정까지는 수월하였으나, 모바일에 제작한 dashboard를 띄우는 것이 불가능하였다. 인터넷 상에서 올라온 방법과 동일하게 시도해보았으나, 모바일에서 dashboard 주소를 입력하면 페이지 오류가 발생하였다. 따라서 본래 계획이었던 Node-RED 프로그램을 이용하여 dashboard를 제작하고 모바일에서 띄우는 방향으로 대체하였다.

## <계획 대비 변동 사항>

### 1) 커튼 기능 제거

본래 수행 계획은 회전 센서 2개를 이용하여, 커튼 양쪽에 센서를 각각 하나씩 설치하는 것이었다. 센서를 동작시켜 원하는 각도만큼 회전시켜 커튼의 개폐를 조절함으로써, 커튼을 원하는 대로 자유자재로 컨트롤 할 수 있도록 하는 것이 목표였다.

하지만 구매한 회전 센서는 원하는 만큼 센서의 출력 부분을 회전시키는 것이 아니라, 원점으로부터 열

마만큼 회전하였는지 각도로 인식하도록 하는 센서였다. 인터넷상에서 우리가 원하는 기능을 가진 회전 센서를 구하기는 매우 어려웠고, 이미 구매한 센서를 적극적으로 활용해보자는 뜻으로 커튼 기능을 제거하고 회전 센서는 가스 밸브에서 터치 센서 대신 사용하는 용도로 변경하였다.

## 2) 습도 센서 및 LCD Display 사용 제외

본래 계획은 DHT22 습도 센서와 LCD Display를 사용하는 것이었다. 먼저 DHT22는 대략 1주의 기간 동안 센서를 작동시키기 위해 시간을 들였으나, 코드상에서 센서가 데이터를 읽어들이기 위해 ready하는 단계에서 작동하지 않았다. 이에 따라 아두이노에서 동작시켜본 결과, 센서가 불량인 것을 확인하였고 DHT11로 센서를 교체하였다.

DHT11에서도 대략 1주 조금 안되는 기간 동안 노력을 기울였다. 하지만 센서를 읽어들이고 난 뒤에 checksum을 하는 과정에서 오류가 발생하여 제대로 동작시킬 수 없었고, 시간 문제상 습도 센서는 먼저 다른 기능을 모두 구현한 뒤에 시도해보기로 하였으나 결국 해결하지 못하였다.

LCD Display는 I2C 통신을 기반으로 구현시키는 것이다. 해당 부품도 대략 1주 동안 시간을 들였는데, 각 LCD의 기능마다 명령어 address가 있는 것을 확인하고 이에 맞게 기능을 구현하려 하였다. 하지만 LCD의 light를 on/off 기능까지만 동작 가능하였고, 교수님의 도움을 받아 조금 더 해결해보려 하였으나 원하는 문자를 LCD에 띄우지 못하여 결국 성공하지 못했다.

## 3) 감지 시간 판단 기준 변경

처음에 사람의 취침을 판단하고, 가스 밸브가 열렸는지 판단하는 기준에는 일정 시간 동안 그 상태가 유지되어야 한다는 조건이 있었다. 예를 들어, 사람의 취침을 확인할 때 현재 시각이 sleeping time인지, 온도 센서의 target 온도 조건을 만족하는지 외에도 이 온도가 1분 정도 유지되어야만 LED를 끄도록 하는 것이다. 이런 판단 방식이 신뢰도가 높을 것으로 예상하였지만, 사용자가 침대에 누고 1분이라는 시간을 기다려야 한다는 점이 오히려 불편함을 느낄 수 있다고 생각하게 되었다. 또한 최종 시연에서도 상태 변화를 바로바로 알 수 있게 하는 편이 좋다고 생각하였기에, 최종적으로 일정 시간 동안 유지되어야만 판단하도록 하는 조건을 제거하였다. 또한, 가스 밸브가 열린 것으로 판단하는 기준에도 이 변경 사항을 적용하였다. 이는 가스 밸브가 열려 가스가 누출되면 큰 사고로 이어질 수 있기에 조기에 빠른 대처를 하기 위함이다.

## 4) 취침 시간에 전체 LED light off 기능 한번 수행으로 변경

처음에 자동으로 LED를 끄는 기능(동시에 무드등을 켜는 기능)은 현재 시각을 설정한 sleeping time과 비교하여 sleeping time 여부를 확인하고, 온도 센서의 target 온도를 만족시키면 동작하도록 설계하였다. 이렇게 설계하고 작동시켜보니, sleeping time에 침대에 누워서 모든 LED를 끄고, 침대에 누운 상태에서 Node-RED dashboard로 LED를 조작하려고 하면 자동으로 LED를 끄는 기능 때문에 LED를 켜려고 해도 바로 LED가 꺼지고, 무드등의 경우 침대에서 무드등을 끄고 싶어도 바로 켜지는 문제가 발생하였다. 이는 자동으로 LED를 조작하는 조건인 sleeping time과 target 온도를 지속적으로 확인하기 때문에 발생한 문제였다. 따라서 사용자의 쾌적하고 자율적인 조작을 위해서 sleeping time에 자동으로 LED를 끄는 기능은 최초 1번만 작동하도록 하고, 이 기능이 한 번 작동한 이후에는 사용자의 버튼 조작으로 LED를 제어할 수 있도록 하였다. 그리고 Node-RED dashboard에 ALL OFF라는 버튼을 추가하여 이 버튼 누르는 것으로 간단하게 모든 LED를 끌 수 있도록 설계하였다.

## ② 개발 결과물 목표 성능 지표 달성 여부 (◎△×)

전등	
자동으로 LED를 제어하는 판단 기준이 신뢰도가 있는가?	△
취침 상태로 판단한 후 얼마나 빠르게 LED가 제어되는가?	◎
에어컨/보일러	
에어컨과 보일러의 threshold 온도가 적절한가?	◎
적정 실내 온도가 될 때까지 제대로 작동하는가?	◎
가스 밸브	
센서를 활용해 가스 밸브의 상태를 얼마나 정확히 인식하는가?	◎
가스가 샌다고 판단한 경우, 부저는 잘 작동하는가?	◎
현관	
PIR 센서를 사용한 감지가 정확하게 되는가?	◎
부저 소리는 적당하며 충분한 알림과 범죄예방에 도움이 되는가?	◎
Node-RED(IoT 플랫폼)	
Dashboard에 센서의 값이 잘 출력되는가?	◎
Dashboard에서 버튼을 조작하는 것으로 집 안의 제어가 잘 되는가?	△

\* 성능 지표는 프로젝트 계획서에서 작성한 항목들을 토대로 나열하였다.

\* 위에서 정리한 성능 지표를 토대로 달성률을 계산해보았을 때, 계획 대비 90%를 달성하였다.

$$\Rightarrow \frac{9}{10} \times 100(\%) = 90.0\%$$

## 5. 개발 일정

<초기 개발 일정>

3월 4주	집 모형 설계	5월 3주	보일러/에어컨용 LED, 온습도 센서 보드에 적용 및 코딩
4월 1주	Node-RED 설계 및 dashboard 구성	5월 4주	현관용 부저, 적외선 센서 보드에 적용 및 코딩
4월 2주	Node-RED 설계 및 dashboard 구성	6월 1주	가스밸브용 버튼 센서 보드에 적용 및 코딩
4월 3주	각 센서와 보드 testing	6월 2주	집 모형에 실적용 및 최종 점검
4월 4주(중간고사 기간)	각 센서와 보드 testing	6월 3주(결과 발표)	프로젝트 결과 발표
5월 1주(중간 발표)	프로젝트 중간 발표		
5월 2주	전등/무드등/TV용 LED, 열감지 센서보드에 적용 및 코딩		

<실제 개발 일정>

3월 4주	집 모형 설계	5월 3주	PIR 센서, 가스 밸브용 회전 센서 및 부저 보드에 적용 및 코딩 (& 온도 센서 testing)
4월 1주	집 모형 센서 연결도 구상	5월 4주	Node-RED 설계 및 dashboard 구성 (& 온도 센서 testing)
4월 2주	LCD, PIR 센서 & 보드 testing	6월 1주	각 LED, 온도 센서 보드에 적용 및 코딩
4월 3주	습도 센서, 부저 & 보드 testing	6월 2주	집 모형에 실적용 및 최종 점검
4월 4주(중간고사 기간)	온도 센서, 회전 센서 & 보드 testing	6월 3주(결과 발표)	프로젝트 결과 발표
5월 1주(중간 발표)	프로젝트 중간 발표		
5월 2주	온도 센서, 습도 센서 testing		

**4월 1주차** : 초기에 Node-RED를 설계하려고 하였으나, 보드 연결선이 도착하지 않은 관계로 보드와의 통신을 확인할 수 없어서 집 모형에 센서 및 LED의 위치를 구상하는 것으로 변경하였다.

**4월 2주차** : Node-RED를 설계하는 것보다 각종 센서와 LCD의 testing이 중요하다고 생각하여 초기 예정보다 센서와 보드의 testing 기간을 더 늘리고, Node-RED 설계 예정을 뒤로 미루게 되었다.

**5월 2주차** : 센서 testing 과정 중 온도/습도 센서의 값을 읽어오지 못하는 문제가 발생하여 자세한 검증을 해보고자 센서 testing 기간을 1주일 늘리게 되었다.

**5월 3주차** : 온도 센서의 값을 읽어오지 못하는 문제를 아직 해결하지 못하여 온도 값이 필요한 에어컨/보일러 및 취침 시 LED를 제어하는 부분은 뒤로 미루고 다른 부분을 먼저 진행하였다.

## 6. 기대 효과

- 임베디드와 IoT 플랫폼을 이용하여, 자택의 상황을 모바일로 확인하고 손쉽게 컨트롤 할 수 있는 홈케어 서비스 시스템을 개발할 수 있다.
- 취침을 센서 하나로 판단한다는 점에서 비슷한 기능을 제공하는 타 서비스보다 더 적은 cost 소모로 최적의 효율을 낼 수 있는 시스템을 개발하도록 한다.
- 홈케어 서비스를 개발함으로써 앞으로 더욱 발전할 홈케어 플랫폼에 대해 발전 가능성을 도모할 수 있다.

## 7. 향후 계획

- 지금까지 진행했던 프로젝트에 더불어, 라즈베리파이 보드를 이용하여 진행한 프로젝트를 더욱 확장하는 방향으로 나아갈 것이다.
- 라즈베리파이 보드를 이용하여 이번 프로젝트에서 구현하지 못한 센서와 새로운 센서(PM센서, 지문인식 센서 등)를 추가하여 집의 완성도를 높일 계획이다. 현관에 지문인식 센서를 설치하고 인식한 지문을 이용하여 현관문을 열어주는 기능이나 원격관리를 할 수 있는 어플리케이션을 제작할 계획이다.



- 우선 현실적으로 진행 가능한 프로젝트를 구상해보고 있으며, 아직 확정된 계획이 아니므로 추후에 변동될 가능성은 있다.