임베디드소프트웨어001\_자유과제보고서\_12161575박재형

컴퓨터공학과 12161575 박재형

**개요**

이번과제는 자유과제로 지금까지 배운 것을 이용해서 주제를 정하고 구현하는 과제입니다. 임베디드 시스템은 일상 생활에 쓰이는 많은 장치들에 사용되고 있습니다. 특정 목적을 가진 시스템이고 실생활에 적용해서 이번 주제를 정하고 싶어서 히터를 구현했습니다. 히터의 모든 기능을 구현할 수는 없었지만 기본적인 히터의 기능을 구현했습니다.

**구상**

**기능**

1. 현재 온도를 측정하고 FND 왼쪽에 현재 온도를 출력합니다.

2. FND 오른쪽에 희망온도를 출력합니다. 희망온도(20~35)

3. 스위치1을 누르면 희망온도가 1 상승합니다.

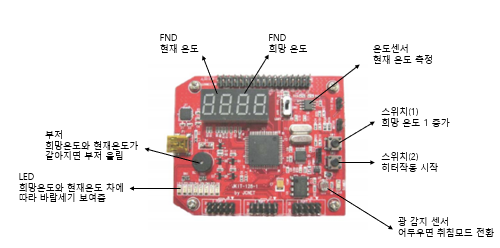
4. 희망온도가 현재온도보다 높을 경우 스위치 2를 누르면 히터가 작동합니다.

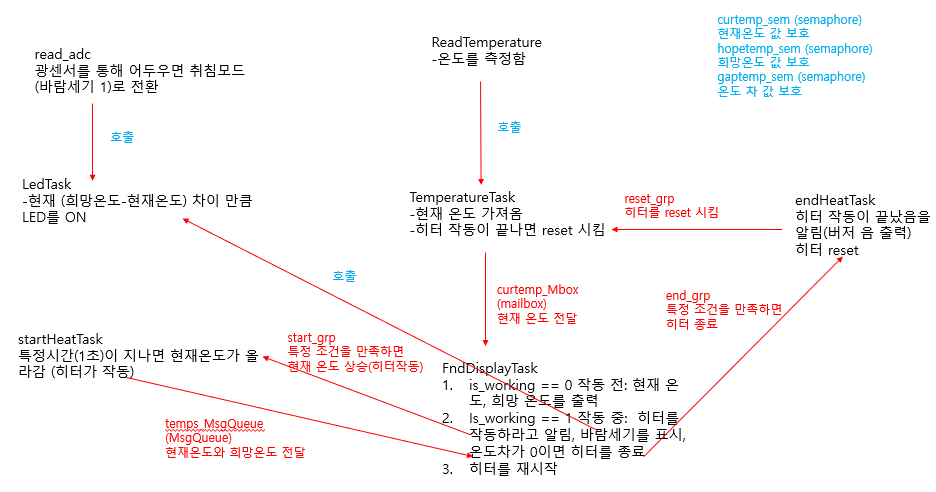
5. 1초가 지나면 현재온도가 1씩 증가합니다. (히터가 작동하는 것을 표현)

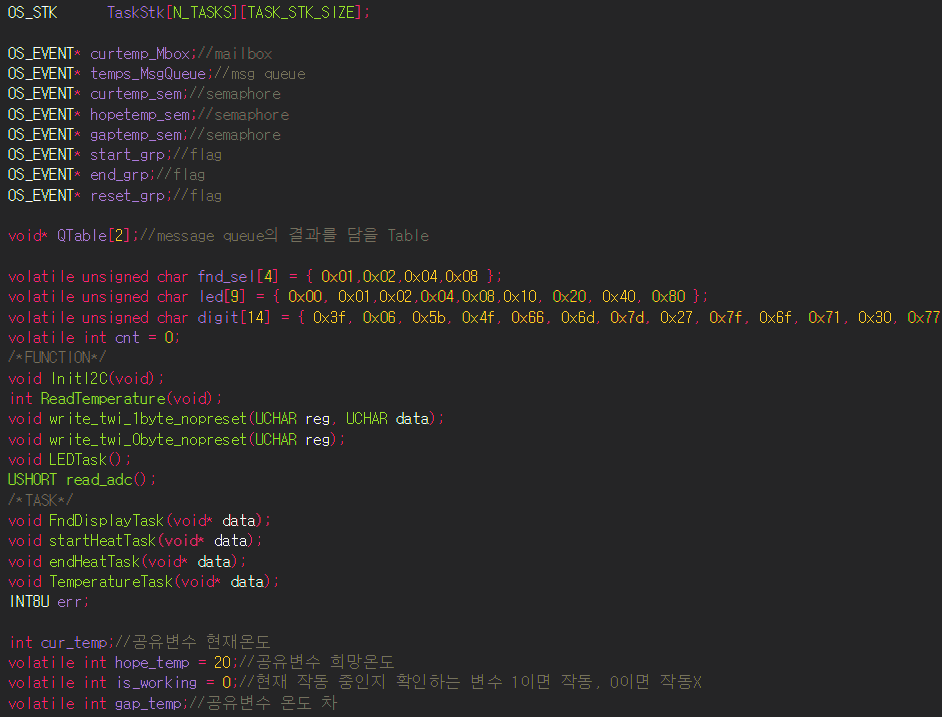
6. 희망온도 – 현재온도 한 차이만큼 LED에 표시합니다. (바람세기를 표현)

7. 어두워지면 바람세기를 1로 바꿉니다. (야간모드를 표현)

8. 현재온도와 희망온도가 같아지면 부저가 울리고 히터가 reset이 됩니다.

**하드웨어:**

**소프트웨어:**

**구현**:

변수

curtemp\_Mbox; 현재온도를 전달하기 위한 mailbox

temps\_MsgQueue: 현재온도와 희망온도를 전달하기 위한 msg queue

curtemp\_sem: 현재온도를 보호하기 위한 semaphore

hopetemp\_sem: 희망온도를 보호하기 위한 semaphore

gaptemp\_sem: 온도 차를 보호하기 위한 semaphore

start\_grp: 시작을 알리기 위한 flag

end\_grp: 끝을 알리기 위한 flag

reset\_grp: reset을 하기 위한 flag

void\* QTable[2]: message queue의 결과를 담을 배열

volatile int cnt = 0: 현재온도를 시작하기전에 한 번만 측정하기 위해서 만든 변수

int cur\_temp: 공유변수 현재온도

volatile int hope\_temp = 20: 공유변수 희망온도 default 값은 20

volatile int is\_working = 0: 현재 작동 중인지 확인하는 변수 1이면 작동, 0이면 작동X

volatile int gap\_temp: 공유변수 온도 차

함수

InitI2C: 온도센서를 초기화

ReadTemperature: 온도를 측정해서 반환

write\_twi\_1byte\_nopreset: 9비트, Normal

write\_twi\_0byte\_nopreset: Temp Reg 포인팅

LEDTask: 바람세기를 LED에 표시하는 함수

read\_adc: 현재 빛 값 읽어오는 함수 (analog -> digital)

TASK

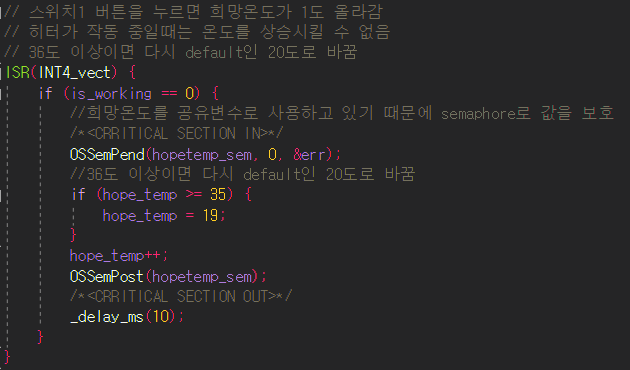
FndDisplayTask: FND에 현재온도와 희망온도를 출력하는 Task

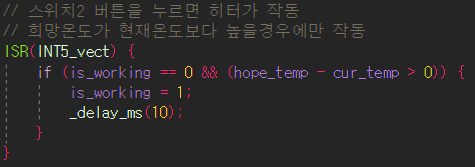
startHeatTask: 히터를 작동시키는 Task

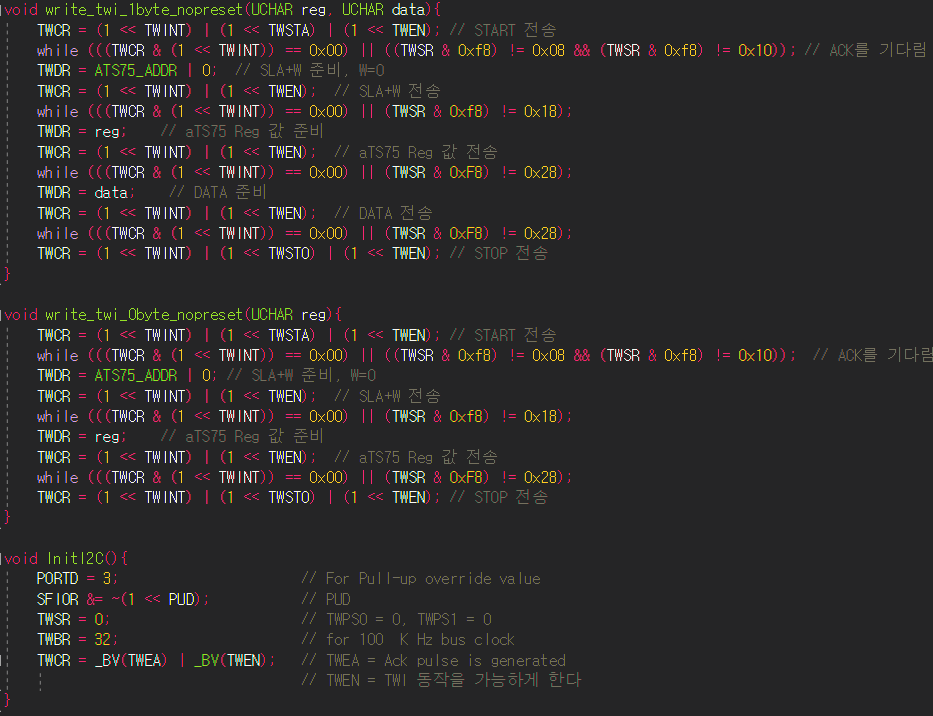
endHeatTask: 히터 작동이 끝났음을 알리는 Task

TemperatureTask: 현재온도를 읽어오는 Task

사용할 센서와 Event 객체를 초기화합니다. Task를 생성하고 우선순위를 지정합니다.

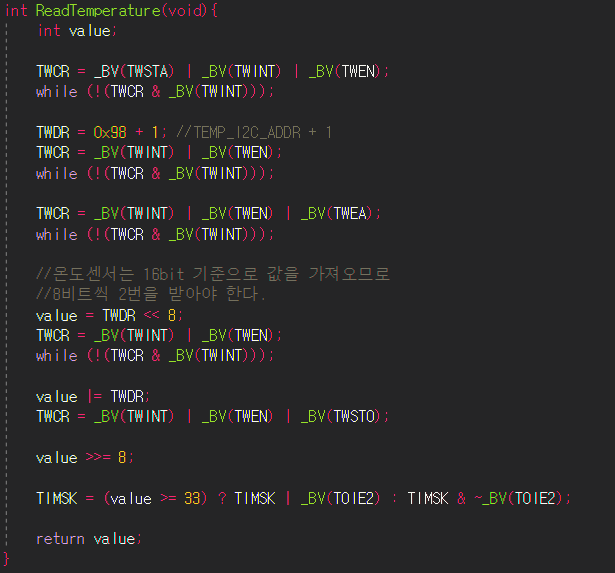
스위치 1을 누르면 희망온도가 1도 올라갑니다. 작동 전에만 희망온도를 설정할 수 있습니다. 희망온도가 공유변수이고 값을 조정하기 때문에 semaphore를 이용해서 값을 보호합니다. 36도 이상이면 default값인 20도로 다시 설정합니다.

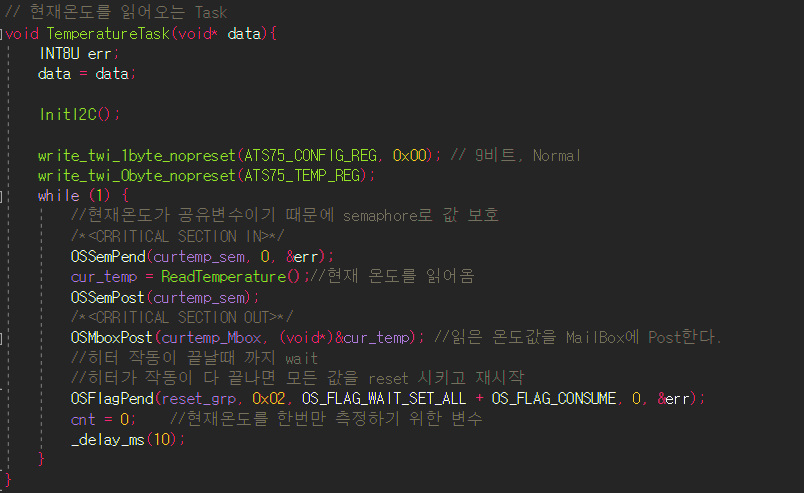
스위치 2를 누르면 히터를 작동시킵니다. 이때 희망온도가 현재온도보다 높을 경우에만 히터를 작동시킵니다.

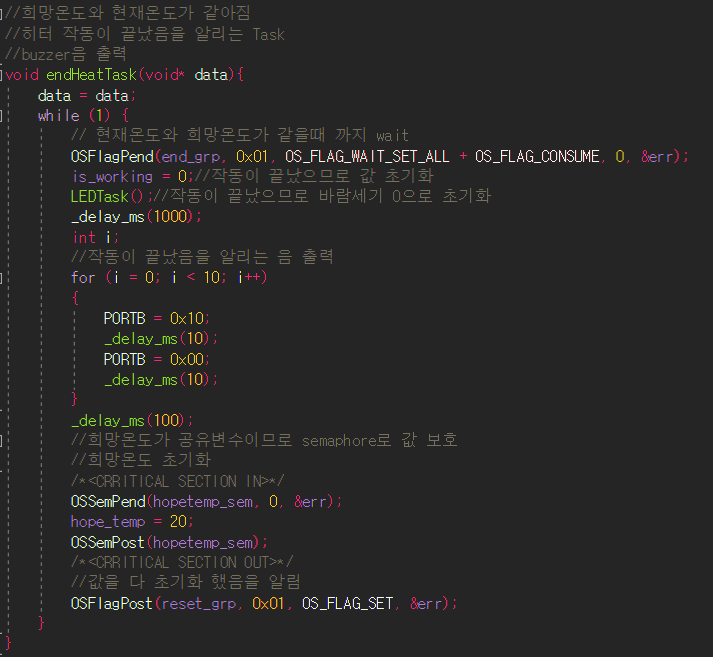
write\_twi\_1byte\_nopreset: 9비트, Normal

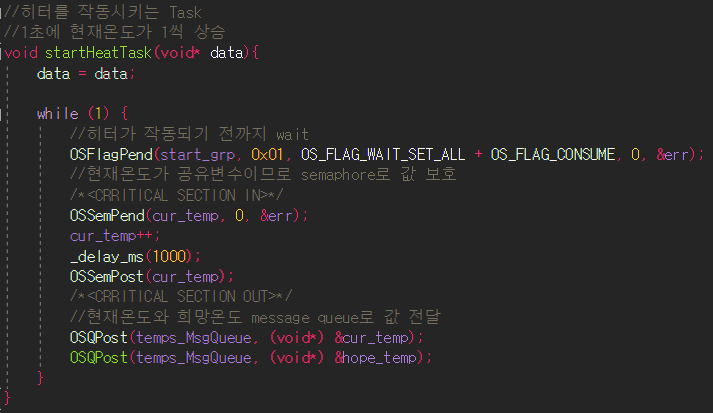
write\_twi\_0byte\_nopreset: Temp Reg 포인팅

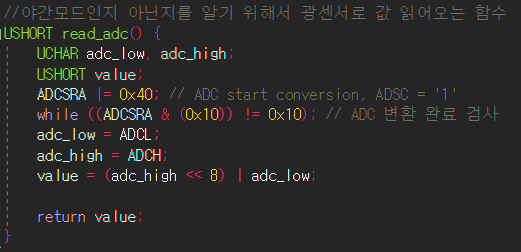
InitI2C: 온도센서를 초기화

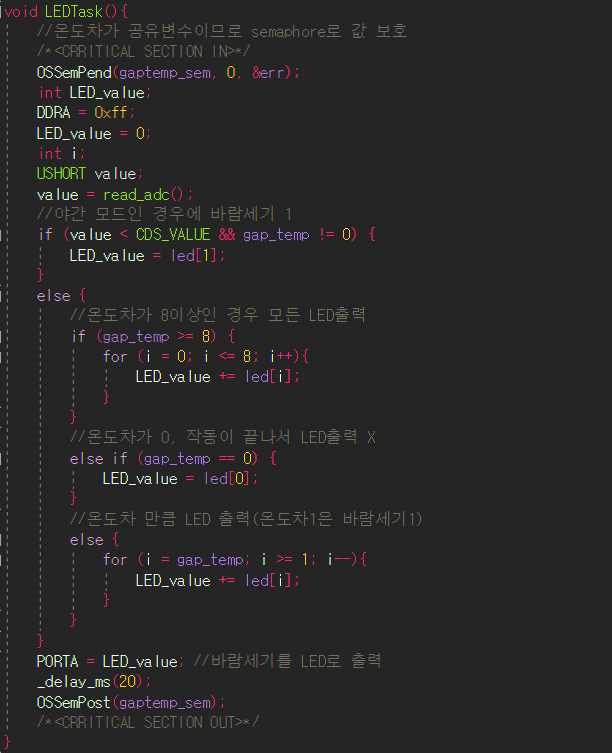
현재온도를 측정하는 함수

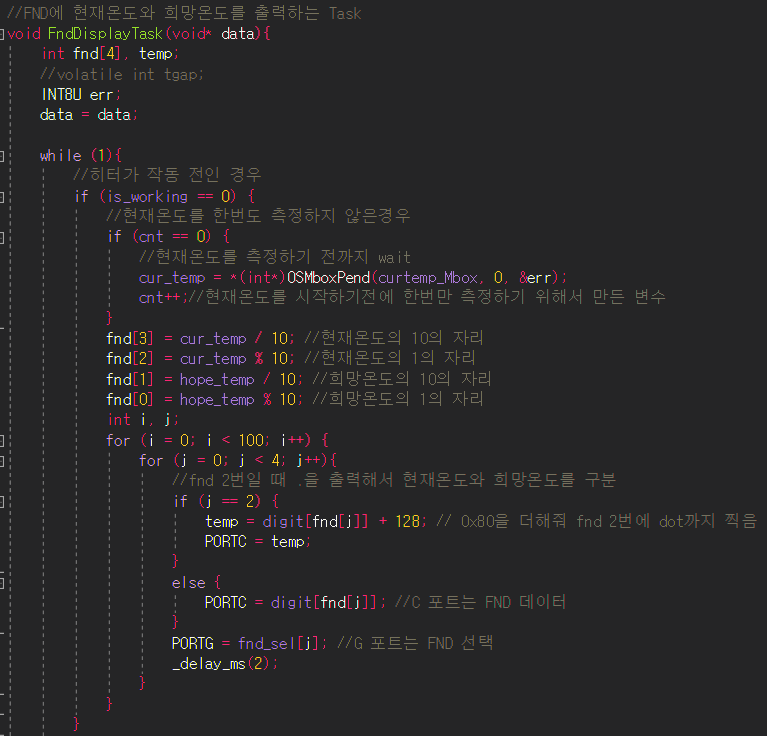
현재온도를 읽어오는 Task입니다. 현재온도가 공유변수이기 때문에 semaphore로 값을 보호합니다. cur\_temp에 현재온도 값을 저장하고 curtemp\_Mbox(mailbox)로 온도 값을 post합니다. 히터가 작동이 끝나면 reset을 해야 하기 때문에 reset flag의 0x02 값이 설정될 때까지 wait을 하게 됩니다. 0x02값이 설정이 되면 다시 현재온도를 측정합니다. (히터를 reset하고 다시 시작한다는 의미입니다.)

히터 작동이 끝났음을 알리는 Task입니다. 현재온도와 희망온도가 같아질 때까지 wait을 하게 됩니다. end\_grp를 통해 0x01값이 set되면 작동이 끝났으므로 is\_working 값을 0으로 초기화합니다. 작동이 끝났으므로 바람세기를 0으로 초기화합니다(LEDTask 이용). 작동이 끝나서 부저음을 울려서 알립니다. 희망온도를 20도로 다시 초기화합니다. 이때 공유변수를 사용하기 때문에 semaphore를 이용해서 값을 보호합니다. reset을 했다는 것을 reset\_grp를 통해 알립니다. Reset\_grp 값을 0x01로 set하고 post 합니다.

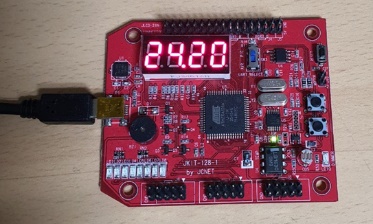
히터를 작동시키는 Task입니다. 히터가 작동되기 전까지 wait을 합니다. start\_grp를 통해 기다립니다. 현재온도가 공유변수이기 때문에 semaphore로 값을 보호합니다. 히터가 작동하는 것을 표현한 것이기 때문에 1초가 지나면 현재온도를 1도 증가시킵니다. Message queue를 통해 현재 온도와 희망온도를 post합니다.

야간모드인지 아닌지를 알기 위해서 광센서로 현재 빛 값을 읽어오는 함수입니다. Analog 값을 digital로 변환합니다.

현재 바람세기를 출력하는 함수입니다. 온도 차가 공유변수 이므로 semaphore를 이용해서 값을 보호합니다. 현재 빛의 값을 읽어와서 어둡다고 판단이 되면 야간모드 이기 때문에 바람세기를 1로 설정합니다. 온도차가 8이상인 경우에는 모든 LED를 출력합니다. 온도 차만큼 LED를 출력합니다. 온도 차 1은 바람세기 1 입니다.

FND에 현재온도와 희망온도를 출력하는 Task입니다. Is\_working이 0인경우는 작동 전인 경우입니다. Cnt가 0이면 현재온도를 한 번도 측정하지 않은 경우이므로 curtemp\_Mbox를 통해서 현재온도를 불러옵니다. fnd에 현재온도와 희망온도를 출력합니다. 구분하기 위해서 .을 찍습니다.

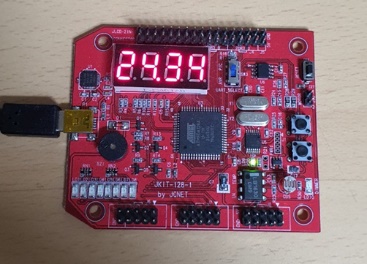
is\_working = 1인 경우는 히터가 작동 중이라는 의미입니다. Start\_grp Flag로 히터를 작동하라고 알립니다. Msg\_queue로 현재 온도와 희망온도를 받아올 때까지 wait을 합니다. 공유변수인 온도 차를 구하기위해 semaphore를 이용해서 값을 보호합니다. 온도차가 0인 경우에는 히터를 종료시키라고 알립니다. End\_grp값을 0x01로 set하고 post해서 알립니다. 재시작을 하기 위해 reset\_grp를 통해서 0x01 값이 set 될 때까지 wait을 하고 다시 재시작 하기 위해서 현재 온도를 측정하는 task로 reset\_grp 0x02를 통해 알립니다. 온도차가 0이 아닌 경우에는 LEDTask를 통해서 바람세기를 LED에 표시합니다. 그리고 현재온도와 희망온도를 FND에 출력합니다.

**구현결과:**

초기상황

현재온도: 24도

희망온도: 20도



희망온도 설정

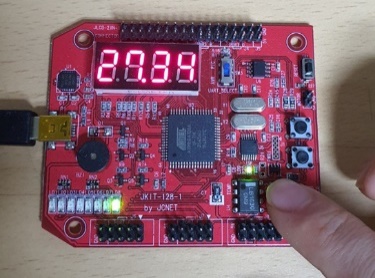
현재온도: 24도

희망온도: 34도

히터가 작동중 (1초에 현재온도 1씩 증가)

온도차이가 8이상이므로 모든 LED 출력

현재온도: 25도

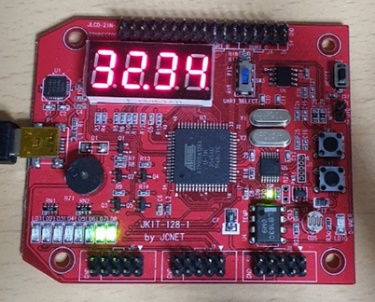
희망온도: 34도

야간모드 (1초에 현재온도 1씩 증가)

온도차이가 7이지만 야간모드 이므로 LED는 1만 출력(바람세기 1)

현재온도 27도

희망온도 34도

히터가 작동중 (1초에 현재온도 1씩 증가)

온도차이가 2이므로 LED 2개 출력

현재온도 32도

희망온도 34도

작동이 끝나서 부저 울리고 reset

현재온도 24도 (측정 값)

희망온도 20도 (default)

**데모영상에 대한 설명:**

처음 부분은 SW1을 눌러서 희망온도를 설정하는 부분을 보여줍니다. 36도를 넘어가면 다시 default 값인 20도로 설정합니다. 그 다음 부분은 sw2를 눌러서 히터를 작동하는 부분을 보여줍니다. 온도차가 나는 만큼 LED에 바람세기를 표시합니다. 이 때 광센서로 어둡다고 느끼면 야간모드로 전환합니다. 야간모드는 바람세기 LED1을 표시합니다. 히터가 작동하는 것은 1초에 현재온도가 1씩 증가하게 표현했습니다. 현재온도와 희망온도가 같아지면 히터가 작동을 끝내고 부저를 울립니다. 그리고 다시 현재온도를 측정하고 희망온도를 default 값인 20도로 reset합니다.

**한계점과 보완할 점:**

주제를 계획하는 초반에 switch1은 희망온도를 상승시키고 switch2는 희망온도를 감소시키고 아무런 동작을 하지 않고 2초이상 지나면 자동으로 히터가 동작하게 하려 했습니다. 그러나 아무런 동작을 하지 않고 2초이상 wait후 작동하는 부분에 구현의 어려움으로 구현하지 못했습니다. 또한 광센서를 이용해서 바람세기를 표현할 때 매 순간마다의 값을 읽어 오는 것이 아니라 현재온도가 상승하는 시점에서 그 때의 광 센서 값을 읽어 오기 때문에 실시간성을 완벽히 지키지 못하는 부분을 보완해야 할 것 같습니다. 또한 히터가 작동하면서 온도가 상승할 때 FND값들이 매끄럽게 넘어가지 않는 부분이 있는데 이 부분을 조금 더 보완해야할 것 같습니다.