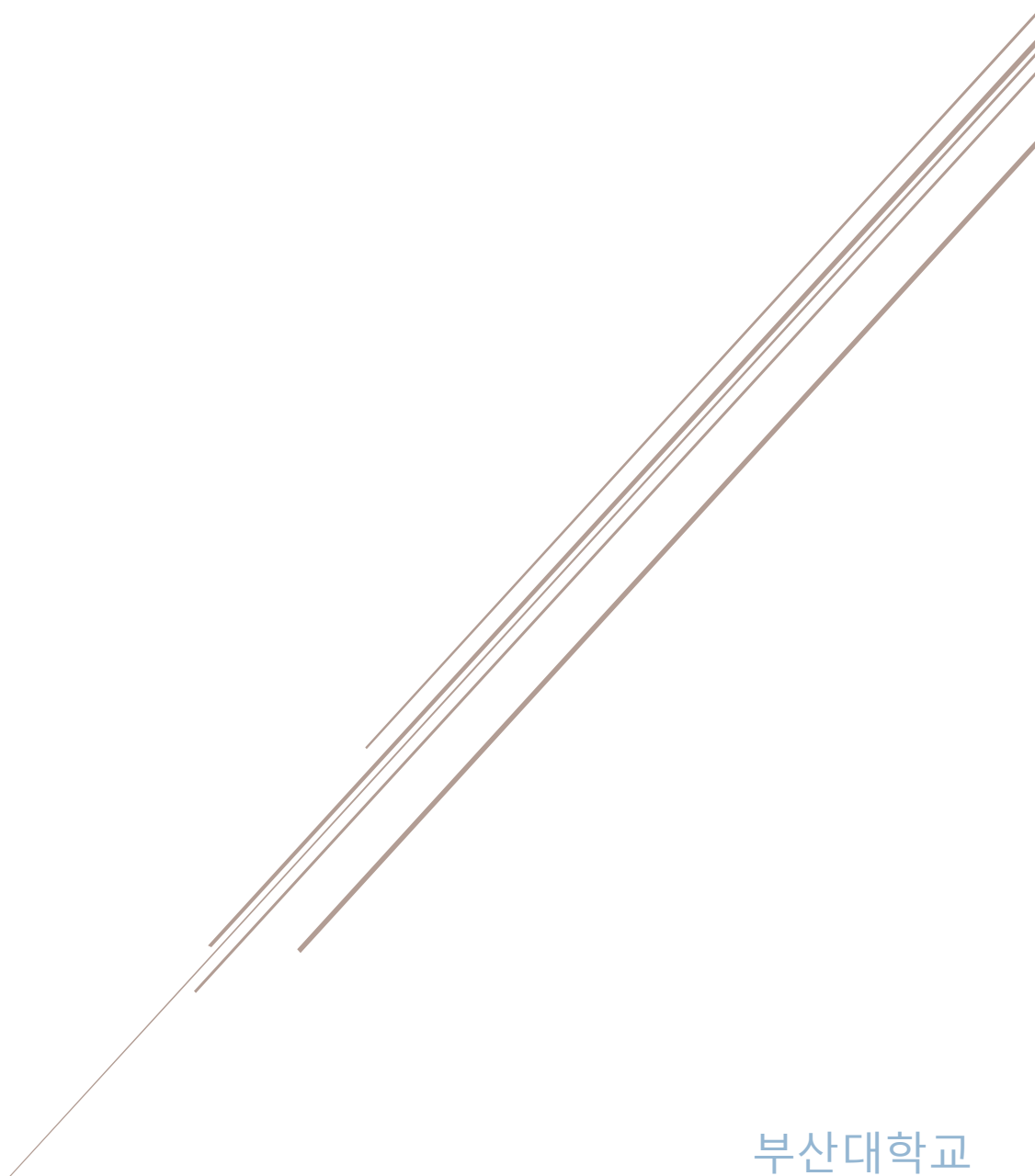


# 팀 프로젝트 최종 보고

이승윤, 심재영



부산대학교

임베디드시스템 (060)

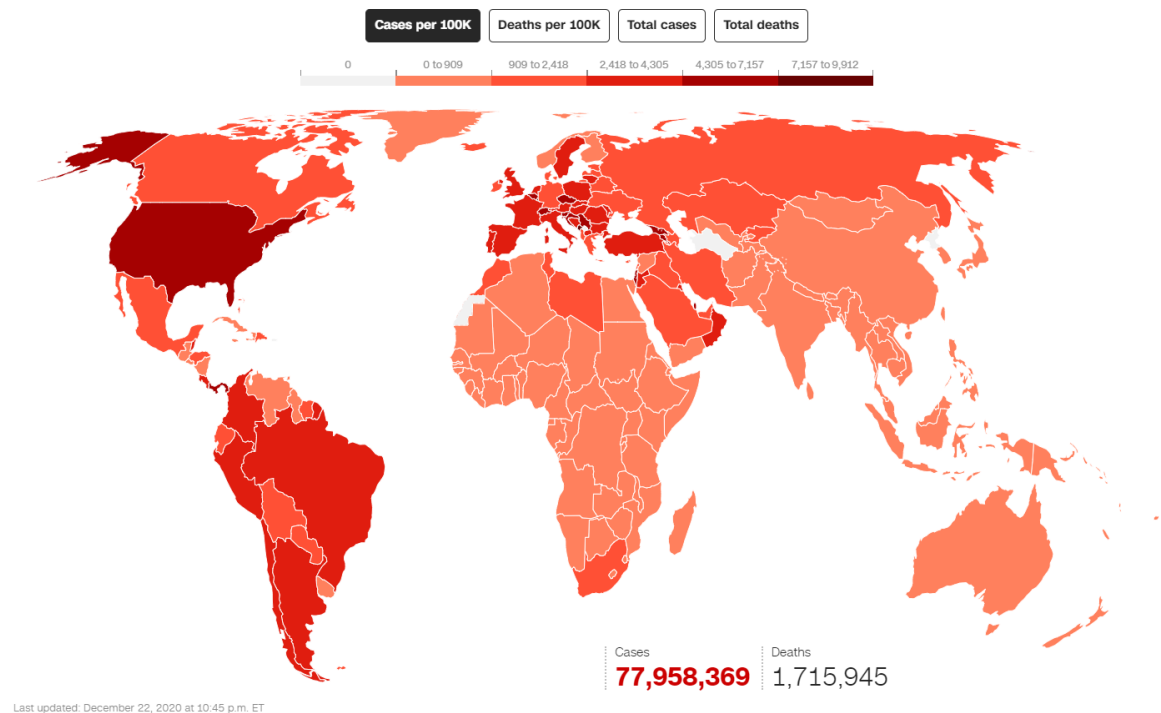
## 목차

개발 시스템 개요 .....	2
개발 배경 .....	2
기존 시스템 .....	3
H-TEC .....	3
배경 지식 .....	5
H/W 구성 .....	6
센서 .....	6
개발 환경 .....	6
기능 .....	7
주요 기능 개요 .....	7
RTos 포팅 .....	8
초기화 .....	9
감지 .....	9
온도 측정 .....	10
알림 .....	11
결과 .....	12
역할 분담 .....	12
주요 일정 .....	12
요약 .....	12
동작 .....	13
참고 문헌 .....	14

## 개발 시스템 개요

### 개발 배경

2020 년 코로나바이러스감염증(COVID-19)으로 인해 전세계적으로 많은 사망자와 감염자가 발생했고 지금까지도 계속 피해가 이어지고 있는 상황이다. 세계적으로 인명 피해 뿐만 아니라 부수적인 경제적 피해가 발생하는 등, 세계가 안타까운 상황 속에 놓여 있다.



[1]

확진자와의 접촉을 예방하기 위해 현재 사람들이 어떤 공간에 모이게 되는 경우, 온도 체크를 통한 온도 측정 이후에 정상 체온일 경우에 입장이 가능하도록 조치를 취하고 있다. 실제로 '온도측정'은 코로나 19 에 감염되어 발열 증상이 있는 사람(즉, 정상 체온보다 높은 사람)을 감지하는데 효과적이라고 알려져 있다. [2]

## 기존 시스템

요즘은 식당이나 건물 내부에 들어가게 될 때 발열 확인이 일상화되었다. 발열 확인 방법에는 측정자가 직접 측정 기구를 이용하여 발열을 확인하는 방법과 측정 기구를 비치해두고 피측정자가 직접 발열을 확인하도록 하는 방법이 있다. 대부분의 소규모 업장에서는 종업(또는 안내)원이 직접 발열 체크를 하는 경우를 자주 볼 수 있다. 이때 측정을 위해 측정자와 피측정자가 밀접 접촉하게 되는 문제점이 있다. 측정 기구를 비치하는 경우에는 다른 사람이 사용하던 측정 기구를 이용하여 확인하여 감염 위험이 있으며 사람들이 많이 드나드는 대규모 빌딩 같은 곳의 카메라와 온도 센서 등을 장착한 대형 장치는 지키는 사람이 없으면 발열 확인 없이 지나가는 경우가 많은 문제점이 있다.

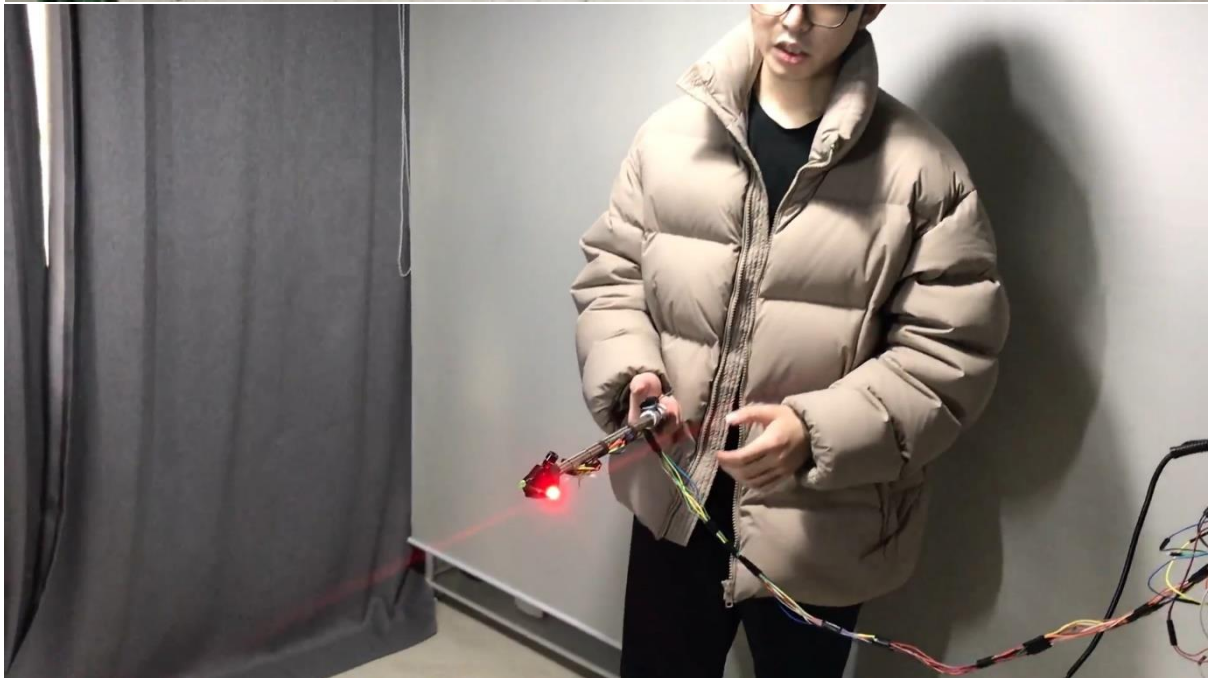
## H-TEC

이러한 문제 해결을 위해 H-TEC(High Temperature Entrance Checking Technology)을 제안한다. H-TEC은 온도를 측정하여 기준 온도 내의 사람이 통과할 수 있도록 하는 기본적인 온도 측정 기능과 함께, 온도를 측정하는 사람과 출입을 하고자 하는 사람 사이에 혹시라도 있을 접촉을 예방하고 거리두기를 실천하기 위한 임베디드 시스템이다.

주로 측정자가 직접 접촉식 온도계를 이용하는 소규모 업장을 대상으로 설정하고 접촉식 온도계와 대형 고정식 장치의 중간 정도 성능을 목표로 한다. 비접촉식 적외선 온도 센서를 이용하여 여러 사람을 측정해도 오염되지 않도록 하고 직원이 직접 발열을 확인하는 특성을 고려하였을 때 직원과 측정자 간의 접촉을 최소화할 수 있도록 긴 몸체를 사용한다.

온도를 측정하는 사람은 터치 센서를 눌러 정확한 온도를 측정할 수 있다. 측정된 사람의 온도를 바탕으로 부저 및 LED를 이용하여 적정 범위 내의 온도인지 확인하고 사용자에게 이를 알려 원활하고 안전한 발열 확인 작업 수행이 기대된다.

ARM Cortex-M3가 부착된 ST사의 STM32F100RB 마이크로 컨트롤러 유닛이 장착된 보드를 사용하며 실시간 운영 체제인 uC-OS2를 탑재하여 각 센서를 제어한다. 이를 통해 임베디드 시스템에 대한 이해와 설계 능력을 배양하고 수업에서 들은 개념, 특히 ARM 프로세서를 활용한 개발 방법 및 실시간 운영 체제에 대해 심화적으로 학습할 수 있다.



## 배경 지식

Cortex 는 ARMv7 의 별칭으로 A, R, M 의 3 가지 프로파일로 나뉜다. M 프로파일은 마이크로 컨트롤러 및 저비용 어플리케이션 동작에 적합하며 thumb2 명령어를 독립적으로 사용할 수 있다. context switch 발생시 일부 레지스터를 하드웨어적으로 스택에 저장/복구할 수 있도록 한다. C 로 프로그래밍 가능하며 스택 기반 예외 처리 모델을 가지고 있다. Cortex-M3 의 CM3 코어는 하바드 구조 3 stage 파이프라인을 보유하고 있으며 NVIC 를 통해 우선 순위 기반 인터럽트를 지원한다. Cortex-M3 는 블루투스, GPS 등 다양한 모듈에 사용된다.

실시간 운영체제는 계산의 정확도 외에 CPU 의 시간 관리의 정확도가 중요한 시스템을 위한 운영체제이다. 선점형 스케줄링 방식(또는 이벤트 구동 방식)과 시분할 스케줄링 방식이 있다. 우선 순위 기반의 선점형 스케줄러가 보편적으로 사용되며 선점형 스케줄러는 우선 순위가 높은 작업(task)이 준비(ready)되면 현재 수행 중인 작업을 중단하고 우선 순위가 높은 작업을 수행한다. 특히 응용 프로그램의 우선 순위가 시스템 프로그램의 우선 순위를 넘어설 수도 있을 정도로 프로그래머에 더 큰 프로세스 우선 순위 제어 권한이 주어진다. 시분할 스케줄러는 클럭 인터럽트나 라운드 로빈 같은 주기적 이벤트가 발생할 때 작업(task)의 전환이 일어나며 실제 필요한 것보다 많은 작업 전환이 발생하여 더 자연스럽게 예측하기 쉬운 멀티태스킹을 제공한다.

프로젝트에서 사용할 마이크로 OS2 는 선점형 스케줄러를 사용하는 실시간 운영체제로 간단한 구조를 가지며 크기가 작다. 여러 태스크가 특정 자원에 대해 경쟁 상태에 있다면 무조건 우선 순위가 높은 측이 자원을 독점하는 특징이 있으며 메모리 관리 기능이 있으나 내부적으로 동적인 메모리 관리는 이루어지지 않는다. 운영 체제가 가져야할 최소한의 기능들로 구성되어 있으며 옵션으로 일부 기능을 제거할 수 있다.

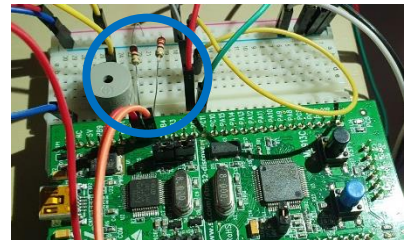
## H/W 구성

### 센서

적외선 온도 센서	터치 센서	부저 / LED
		

H-TEC 시스템은 적외선 온도 센서, 터치 센서, 부저, LED 를 이용하여 구성한다. 사용자가 터치 센서를 누르면 적외선 온도 센서를 이용하여 온도를 측정하고 온도가 적정 범위보다 낮은 경우, 적정 범위인 경우, 적정 범위보다 높은 경우에 대하여 부저 및 LED 를 조작한다.

적외선 온도 센서는 I2C 통신을 이용하여 온도를 읽어온다. SCL, SDA 포트에 연결이 필요하다. slave address 는 0x3A 로 7bit 주소를 사용하며 클럭 주파수를 100KHz 이하로 해야한다. 중요한 점은 SCL 과 SDA 포트를 연결할 때 3.3V 풀업 저항을 연결해주어야 한다. 전송 받을 데이터는 2Byte 로 별도의 온도 변환 계산 과정이 필요하다.



### 개발 환경

STM32 Value line discovery 는 STM32 evaluation 보드로 ST-link 를 내장하고 있다. STM32F100RB 마이크로 컨트롤러 유닛과 Cortex-M3 가 장착되어 있으며 GPIO 포트 및 USART, SPI, I2C, ADC, TIM 을 지원한다. 다양한 개발 도구를 사용할 수 있으며 이번 프로젝트에서는 IAR Embedded Workbench 를 사용한다.

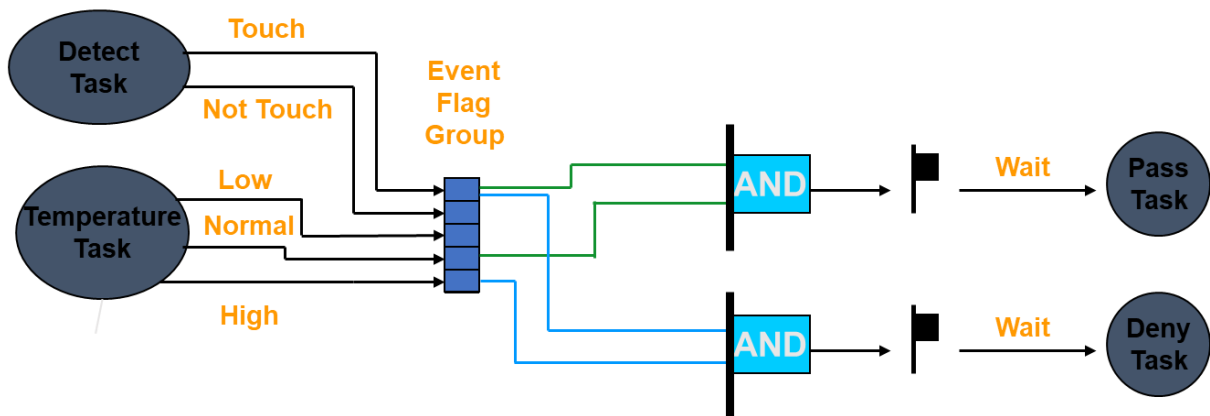




## 기능

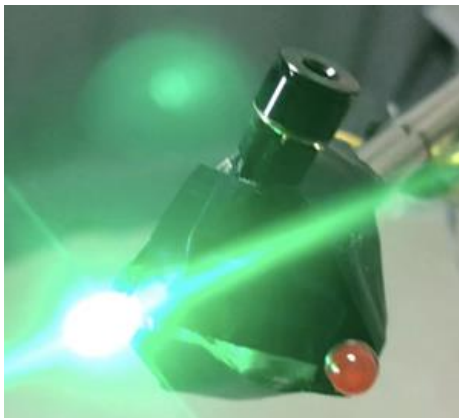
### 주요 기능 개요

기능은 크게 감지, 온도 측정, 알림으로 나눌 수 있으며 감지 작업을 수행하는 detect task 와 온도 측정 작업을 수행하는 temperature task, 알림 작업을 수행하는 pass task, deny task 가 주요 task 이다. 이외에 초기화 작업을 위한 start task, 알림 해제 작업을 수행하는 check task 가 존재한다.



총 3 가지 온도 측정의 경우를 나누어 각 상황에 맞게 알림 기능을 구현하였다.

1. 온도가 정상체온 보다 낮은 경우  
=> LED **RED(빨강)** ON, LED **GREEN(초록)** OFF
2. 정상체온 인 경우  
=> LED **RED(빨강)** OFF, LED **GREEN(초록)** ON
3. 고열인 경우(정상체온 보다 높은 경우)  
=> LED **RED(빨강)** ON, LED **GREEN(초록)** OFF, **BUZZER(부저)** ON





## RTOS 포팅

RTOS 중 마이크로 OS-2 를 사용하도록 코드를 변환한다. uOS-2 와 stm32f10x 관련 헤더파일을 include 해주어 운영체제 관련 코드와 MCU 관련 코드를 사용할 수 있게 한다. include 할 파일이 다양하므로 includes.h 에 필요한 헤더 파일을 추가한다. 각 Task 에서 사용되는 스택 공간과 이벤트 플래그, 여러 Task 에서 사용되는 변수를 전역변수로 선언하여 공간을 할당해준다.



메인 함수에서는 OSInit 을 통해 커널을 초기화하고 필요한 태스크를 생성한다. OSStart 를 호출하여 멀티태스킹을 시작한다. 태스크를 생성할 때 각 태스크 함수, 인자, 시작 스택 주소, 우선 순위, id, 스택 주소, 스택 크기, 확장 태스크 블록, 옵션을 인자로 받는다. 인자와 확장 블록은 불필요하므로 NULL 로 처리하고 ID 는 우선 순위와 일치하도록 생성하였다. 또 사용할 버스에 클락을 공급하고 GPIO pin 설정과 채널 구성을 설정한다.

```
OSInit();

os_err = OSTaskCreateExt((void (*)(void *))detectTask,
                        (void *)0,
                        (OS_STK *)&detectTaskStack[TASK_STK_SIZE - 1],
                        (INT8U)TASK_DETECT_PRIO,
                        (INT16U)TASK_DETECT_PRIO,
                        (OS_STK *)&detectTaskStack,
                        (INT32U)TASK_STK_SIZE,
                        (void *)0,
                        (INT16U)(OS_TASK_OPT_STK_CLR | OS_TASK_OPT_STK_CHK));

OSStart();
```



## 초기화

startTask 가 태스크가 아닌 곳에서 수행할 수 없는 초기화 작업들을 수행한다. OS\_CPU\_SysTickInit 를 통해 시스템 틱을 초기화하고 사용할 이벤트 플래그 및 세마포어를 생성한다. 이벤트 플래그는 태스크간 통신 기능을 위해 사용되며 터치 센서를 통한 감지 작업과 적외선 온도 센서를 통한 온도 측정 작업에 따라 통과 작업 또는 통과 거부 작업이 수행된다. 세마포어는 일정 시간 이후 알림 종료를 위해 사용되며 통과 작업 및 통과 거부 작업에서 설정을 위해 사용되고 확인 작업에서 설정 해제를 위해 접근이 필요하기 때문에 세마포어를 설정하여 동시 접근이 불가능하도록 차단한다.

```
static void startTask(void *p)
{
    CPU_INT08U err;

    OS_CPU_SysTickInit();

    // Create Event Flag
    flagGroup = OSFlagCreate(0, &err);

    // Create semaphore
    sem = OSSemCreate(0);

    stopAlert();
    stopNotice();
    #if (OS_TASK_STAT_EN > 0)
        OSStatInit(); /* Determine CPU capacity.
    #endif

    #if ((APP_PROBE_COM_EN == DEF_ENABLED) || \
        (APP_OS_PROBE_EN == DEF_ENABLED))
        App_InitProbe();
    #endif

    while (DEF_TRUE)
    {
        OSTimeDlyHMSM(0, 0, 0, DELAY_TIME); // To run other tasks
    }
}
```

## 감지

detectTask 는 터치 센서를 이용하여 사용자의 입력을 감지한다. 버튼을 사용할 때는 물리적으로 힘을 주어야 하지만 터치 센서는 손가락을 가져다 대는 형태로 부드러운 느낌을 주는 장점이 있다.



터치 센서를 누를 경우 이벤트 플래그를 이용하여 FLAG\_DETECT 를 설정하며 누르지 않을 경우 FLAG\_DETECT\_NOT 이 설정된다. FLAG\_DETECT\_NOT 은 현재 사용되지 않지만 작업 확장 가능성을 두고 그대로 유지하였다.

```
if (val != 0) // when human detected(auto) or touch button(manual)
{
    OSFlagPost(flagGroup, (OS_FLAGS)FLAG_DETECT, OS_FLAG_SET, &err);
}
else
{
    OSFlagPost(flagGroup, (OS_FLAGS)FLAG_DETECT_NOT, OS_FLAG_SET, &err);
}
```

## 온도 측정


temperTask 는 적외선 온도 센서를 이용하여 온도를 측정한다. 적외선 온도 센서는 신체 접촉 없이 원거리에서 온도를 측정할 수 있다. I2C 통신을 사용한다.

구체적으로 start bit 를 전송하고 slave 주소에 라이트 신호(0)을 더한 0x74 를 전송하고 slave 로부터 ACK 신호를 받을 때까지 대기한다. ACK 신호가 오면 대상 온도를 받아오는 명령인 0x07 을 전송하고 슬레이브가 다시 ACK 응답을 받을 때까지 대기한다. 다시 스타트 비트를 전송하고 이번에는 센서에서 값을 읽어올 것이므로 slave address 에서 read 신호 (1)을 더한 0x75 를 전송한다. 슬레이브에서 ACK 신호가 오면 데이터 바이트를 읽을 수 있다. 그러나 온도 값은 2 바이트이기 때문에 2 번 데이터 바이트를 읽어 합치는 과정이 필요하다. 온도 값을 다 읽어오면 stop 비트를 전송하여 통신을 끝낸다. 100ms 의 대기시간 이후 다시 위 과정을 수행할 수 있다.

가져온 2byte 의 데이터 중 최상위 비트는 에러 플래그이고 에러 플래그가 1 인 경우 바르지 않은 값이 출력되는 경우이다. Raw data 를 섭씨온도로 변환하려면 센서 값에 0.02 를 곱하여 절대온도로 변환한 뒤 273.15 를 빼서 섭씨로 나타낼 수 있다.

```
static int readTemperature()
{
    while (I2C_GetFlagStatus(((I2C_TypeDef *)I2C1_BASE), I2C_FLAG_BUSY))
        I2C_GenerateSTART(((I2C_TypeDef *)I2C1_BASE), ENABLE);
    while (!I2C_CheckEvent(((I2C_TypeDef *)I2C1_BASE), I2C_EVENT_MASTER_MODE_SELECT))
        ;
    I2C_Send7bitAddress(((I2C_TypeDef *)I2C1_BASE), 0x74, I2C_Direction_Transmitter);
    while (!I2C_CheckEvent(((I2C_TypeDef *)I2C1_BASE), I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED))
        ;
    I2C_SendData(((I2C_TypeDef *)I2C1_BASE), 0x07);
    while (!I2C_CheckEvent(((I2C_TypeDef *)I2C1_BASE), I2C_EVENT_MASTER_BYTE_TRANSMITTED))
        ;
    I2C_GenerateSTOP(((I2C_TypeDef *)I2C1_BASE), ENABLE);
    I2C_GenerateSTART(((I2C_TypeDef *)I2C1_BASE), ENABLE);
    while (!I2C_CheckEvent(((I2C_TypeDef *)I2C1_BASE), I2C_EVENT_MASTER_MODE_SELECT))
        ;
    I2C_Send7bitAddress(((I2C_TypeDef *)I2C1_BASE), 0x75, I2C_Direction_Receiver);
    while (!I2C_CheckEvent(((I2C_TypeDef *)I2C1_BASE), I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED))
        ;
    int low = I2C_ReceiveData(((I2C_TypeDef *)I2C1_BASE));
    if (I2C_GetLastEvent(((I2C_TypeDef *)I2C1_BASE) & 0x40 != 0x40)
    {
        int high = I2C_ReceiveData(((I2C_TypeDef *)I2C1_BASE));
        if (high & 0x80 != 0)
        {
            return 20; // Need Delay / return default
        }
        else
        {
            return (high << 8 + low) * 0.02 - 273.15;
        }
    }
    else
    {
        return 20;
    }
}

I2C_AcknowledgeConfig(((I2C_TypeDef *)I2C1_BASE), ENABLE);
}
```



```
temp = readTemperature();
if (temp > high)
{
    OSFlagPost(flagGroup, (OS_FLAGS)FLAG_TEMPER_HIGH, OS_FLAG_SET, &err);
}
else if (temp < low)
{
    OSFlagPost(flagGroup, (OS_FLAGS)FLAG_TEMPER_LOW, OS_FLAG_SET, &err);
}
else
{
    OSFlagPost(flagGroup, (OS_FLAGS)FLAG_TEMPER_NORMAL, OS_FLAG_SET, &err);
}
```

## 알림

앞서 살펴본 이벤트 플래그 작동도에 따라 이벤트 플래그 그룹의 DETECT 플래그와 HIGH 플래그가 조합되면 고열인 상태이므로 RED LED 는 점등하고 부저를 울리게 하여 고온인 상태를 쉽게 구분 가능하도록 하였다. DETECT 플래그와 NORMAL 플래그가 조합되면 적정 온도 범위내에 있는 것으로 GREEN LED 를 점등시켜 상태를 표시한다. 대부분이 적정 범위 내에 속할 것이므로 통과시 부저를 울리게 되면 측정자의 집중력이 분산되고 소리로 인한 피로도가 지속적으로 증가하므로 찾아야할 고열인 상태에서만 부저의 소리가 나도록 구현하였다. 사용자가 터치하지 않아 감지되지 않을 때는 상태 변화가 일어날 필요가 없으므로 별도의 task 를 구현하지 않았고 온도가 낮을 경우에는 계속해서 RED LED 가 점등되어 있기 때문에 측정자는 제대로 측정이 되지 않아 온도를 다시 측정해야 함을 파악할 수 있게 하였다.

```
static void passTask(void *p)
{
    int err;
    while (DEF_TRUE)
    {
        OSFlagPend(flagGroup,
                    (OS_FLAGS)(FLAG_DETECT + FLAG_TEMPER_NORMAL),
                    OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME,
                    100,
                    (INT8U *)&err);
        startNotice();
        stopAlert();

        OSSemPend(sem, 0, (INT8U *)&err);
        if (count == 0)
            count = 1;
        OSSemPost(sem);
        OSTimeDlyHMSM(0, 0, 0, DELAY_TIME); // To run other tasks
    }
}

static void denyTask(void *p)
{
    int err;
    while (DEF_TRUE)
    {
        OSFlagPend(flagGroup,
                    (OS_FLAGS)(FLAG_TEMPER_HIGH + FLAG_DETECT),
                    OS_FLAG_WAIT_SET_ALL + OS_FLAG_CONSUME,
                    100,
                    (INT8U *)&err);
        startAlert();
        stopNotice();

        OSSemPend(sem, 0, (INT8U *)&err);
        if (count == 0)
            count = 1;
        OSSemPost(sem);
        OSTimeDlyHMSM(0, 0, 0, DELAY_TIME); // To run other tasks
    }
}
```

checkTask 는 pass 또는 deny task 에서 LED 및 부저를 작동시키고 일정 시간 이후 이를 reset 하는 역할을 수행한다. 전역변수의 flag 값은 pass 와 deny task 에서도 설정되기 때문에 각 task 에서 전역 변수 값 교체시 세마포어를 사용한다. 각 set, reset 함수를 startAlert, stopAlert, startNotice, stopNotice 로 묶어 헛갈리지 않도록 하였다. 모두 stop 되어있을 경우 기본값이 LED RED ON, LED GREEN OFF, BUZZER OFF 상태로 유지된다.

```
static void stopAlert()
{
    // RED LED
    GPIO_SetBits(GPIOC, GPIO_Pin_12);
    // GREEN LED
    GPIO_ResetBits(GPIOC, GPIO_Pin_11);
    // BUZZER
    GPIO_ResetBits(GPIOB, GPIO_Pin_8);
}

// 통과 거부
static void startAlert()
{
    // RED LED
    GPIO_SetBits(GPIOC, GPIO_Pin_12);
    // GREEN LED
    GPIO_ResetBits(GPIOC, GPIO_Pin_11);
    // BUZZER
    GPIO_SetBits(GPIOB, GPIO_Pin_8);
}
```

```
static void stopNotice()
{
    // RED LED
    GPIO_SetBits(GPIOC, GPIO_Pin_12);
    // GREEN LED
    GPIO_ResetBits(GPIOC, GPIO_Pin_11);
    // BUZZER
    GPIO_ResetBits(GPIOB, GPIO_Pin_8);
}

// 통과 허용
static void startNotice()
{
    // RED LED
    GPIO_ResetBits(GPIOC, GPIO_Pin_12);
    // GREEN LED
    GPIO_SetBits(GPIOC, GPIO_Pin_11);
    // BUZZER
    GPIO_ResetBits(GPIOB, GPIO_Pin_8);
}
```

```
static void checkTask(void *p)
{
    CPU_INT08U err;
    stopAlert();
    stopNotice();
    while (DEF_TRUE)
    {
        if (count != 0)
        {
            check++;
            if (check > 3)
            {
                stopAlert();
                stopNotice();

                OSSemPend(sem, 0, &err);
                count = 0;
                OSSemPost(sem);
                check = 0;
            }
        }

        OSTimeDlyHMSM(0, 0, 0, DELAY_TIME); // To run other tasks
    }
}
```

## 결과

### 역할 분담

<b>이승윤</b>	온도 측정 기능, 알림 기능 운영체제 포팅 및 기타 로직 작성 보고서 작성 동영상 발표
<b>심재영</b>	터치 감지 기능, 버전 관리 장치 디자인 및 세부 설계 자료 조사 동영상 촬영 및 편집

### 주요 일정

항목	일정	기타
프로젝트 계획	2020/11/28 – 2020/12/03	완료
인체 감지 기능	2020/12/04 – 2020/12/16	완료
온도 측정 기능	2020/12/04 – 2020/12/21	완료
알림 기능	2020/12/04 – 2020/12/21	완료 (일부 변경)
운영체제 포팅 및 주요 로직	2020/12/12 – 2020/12/22	완료 (지연)

### 요약

H-TEC 은 기존의 직접 센서를 이용하여 측정자가 온도를 재는 방법을 대체하는 시스템이다. ARM Cortex M3 와 STM32F100RB 마이크로 컨트롤러가 탑재된 보드에서 동작하는 uC-OS2 기반 프로그램을 제작하였으며 시스템은 detectTask, temperTask, passTask, denyTask, checkTask, startTask 를 사용하였다. 터치 센서를 누르면 적외선 센서로 측정된 온도 값을 이용하여 3 가지 상태, 고열인 경우(RED LED, BUZZER), 정상인 경우(GREEN LED), 온도가 낮은 경우(RED LED)에 대해 다른 동작을 수행한다. detectTask, temperTask 가 event flag 를 set 하여 조건을 만족하면 passTask, denyTask 가 pend 하고 있다가 작업을 처리한다. passTask, denyTask, checkTask 는 알림 설정 및 해제를 위해 세마포어를 이용하여 같은 전역변수에 접근한다.

## 동작



시연 영상은 <https://photos.app.goo.gl/mH4dt5wrMuyEEkUN6> 에서 확인할 수 있다.

[1] "World Covid-19 tracker: Latest cases and deaths by country," CNN, 22 12 2020. [온라인].

Available: <https://edition.cnn.com/interactive/2020/health/coronavirus-maps-and-cases>.

[엑세스: 23 12 2020].

[2] 대한민국 정부, "코로나바이러스감염증-19 > 열 스캐너로 코로나 19 환자를 찾아낼 수

있나요?," 보건복지부, [온라인]. Available: :

<http://ncov.mohw.go.kr/shBoardView.do?brdId=3&brdGubun=37&ncvContSeq=2276>. [엑세스:

23 12 2020].