개인 프로젝트 결과 보고서

(3, 4) 반, ( ) 조

조원: (학번: 12141635 ,이름: 최유정)

1. 제목

타이머, LED, 온도 센서, 부저를 이용한 **<과열 방지 세미 오븐>** 구현

2. 목표 시스템

과열 방지 세미 오븐이 가져야 할 기능은 다음과 같다.

1) 타이머를 설정하여 원하는 시간 만큼 오븐을 작동하도록 구동
2) 버튼을 이용하여 시간 설정 및 시작/중지 버튼 구현
3) 작동 중에는 실제 가열 대신 LED 로 가열 여부를 구현
4) 온도 센서를 이용하여 특정 온도를 넘어설 경우 가열을 중지
5) 타이머 종료 시 부저를 이용하여 멜로디 발생

3. 구현에 필요한 기초지식

1) uC/OS II
   - Tasks : 태스크 생성(OSTaskCreate), 태스크 삭제(OStaskdel), 태스크 스택(OS_STK)
   - Synchronization : semaphore, mailbox, Event flag,
2) 디바이스(JKIT -128)
   - 인터럽트 : 버튼 입력 인터럽트, 상승엣지
   - 8비트, 16비트 타이머, 카운터 : 오버플로우 인터럽트, 값비교 인터럽트
   - I2C 통신
   - 입출력(LED, FND, 부저) : 입출력 방향 레지스터, PORT 입출력

4. 시스템 설계

1) TASK

   메인 TASK 1개, 시간 계산을 수행하는 타이머 TASK 1개, 4개의 디바이스(FND, LED, 부저, 온도 센서)의 입출력을 관장하는 디바이스 TASK 4개로 총 6개의 TASK로 구성하였다.

2) 인터럽트

   두개의 버튼 입력을 처리하기 위한 인터럽트(INT4, INT5), 시간을 카운팅 하기 위한 16비트 타이머(TIMER1)의 값 비교 인터럽트, 부저로 출력될 음악 소리의 음계를 주파수로

처리하기 위한 8비트 타이머(TIMER2)의 오버플로우 인터럽트, 총 4개의 인터럽트를 사용한다.

3) 동기화(Synchronization 메소드)

① 온도를 측정하는 TASK가 LED TASK(가열 TASK) 과열 여부를 전달하는 mail box

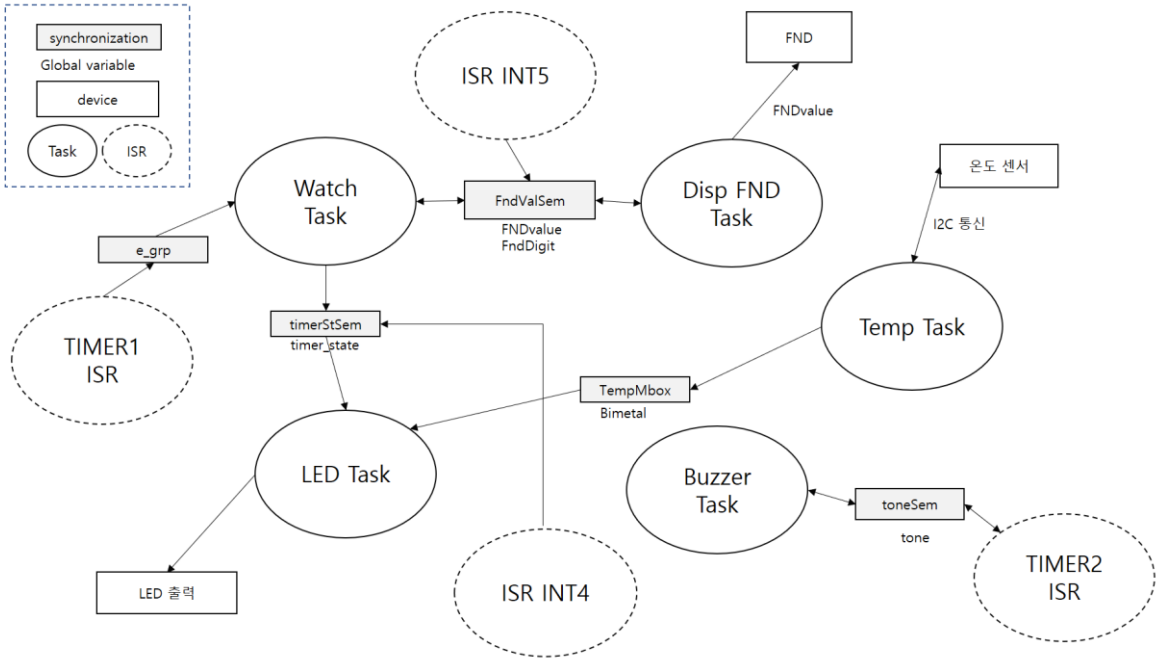
② TIMER1이 0.5초가 지났음을 시계 TIMER에게 알려주는 event flag

③ FND 출력 TASK 와 TIMER TASK 가 서로 같이 읽고 쓰는 FND 출력 값을 저장하는 전역변수를 보호하기 위한 shemaphore

의 3가지 동기화 기법을 사용한다.

4) 전체 프로그램의 구성도

전체 프로그램 구성도 및 흐름도는 다음 도식과 같다.



5. 실제 구현 및 소스 코드 설명

1) 선언부

Task 에 사용되는 스택과 세마포어, 메일 박스, 이벤트 그룹, 동기화 객체 변수를 선언하였다. 또한 task 간 데이터를 교환 할 전역 변수들을 선언하였다. 주요 변수들은 다음과 같다.

```
/*
태스크 스택 : 5개 태스크에 대한 스택구조체이다. Buzzer 태스크는 악보 때문에 크게 잡아놓았다.
*/
OS_STK  LedTaskStk[TASK_STK_SIZE];
OS_STK  TempTaskStk[TASK_STK_SIZE];
OS_STK  DispFndTaskStk[TASK_STK_SIZE];
OS_STK  WatchTaskStk[TASK_STK_SIZE];
OS_STK  BuzzerTaskStk[512];

/*
프로그램에 사용되는 모든 동기화 개체를 선언함.
2개의 세마포어, 1개의 메일박스, 1개의 이벤트 그룹을 사용하였다.
*/
OS_EVENT     *FndValSem;
OS_EVENT     *ToneSem;
OS_EVENT     *TempMbox;
OS_FLAG_GRP  *e_grp;


// FND 디지트 정보와 PORTG를 위한 Selection bit
unsigned char digit[12] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x27, 0x7f, 0x6f, 0x40, 0x00 };
unsigned char fnd_sel[4] = { 0x01, 0x02, 0x04, 0x08 };
/*
   FND에 표시될 값에 대한 전역변수
   DispFnd 태스크는 무한 루프를 돌면서 아래 값을 FND에 출력한다.
   FndDigit는 소수점을 찍을 자리에 대한 비트값을 저장한다.
   0x01 = 맨 우측 FND에 소수점 표시, 0x0F = 모든 자리에 소수점 표시
*/
int FndValue;
unsigned char FndDigit;

unsigned int seconds;       /*타이머 시간을 저장하는 전역변수*/
volatile int timer_state, buzzer_state; // 타이머와 부저의 상태를 저장하는 전역 변수
```

2) main 함수

① OS가 사용할 클럭을 설정한다. 인터럽트가 걸리지 않도록 크리티컬섹션에 들어간다.

② 프로그램에 사용될 세마포어, 이벤트그룹, 메일박스를 생성한다.

③ 버튼 입력에 대한 인터럽트를 설정한다.

④ 각종 디바이스 상태를 초기화 한다.

⑤ BuzzerTask를 제외한 4개의 태스크를 생성한다.

⑥ 멀티태스킹을 시작한다.

```
int main (void)
{
    INT8U    err;

    OSInit();

    OS_ENTER_CRITICAL();
    TCCR0=0x07;
    TIMSK =_BV(TOIE0);
    TCNT0=256-(CPU_CLOCK_HZ/OS_TICKS_PER_SEC/ 1024);
    OS_EXIT_CRITICAL();

    // 2개의 버튼에 대한 인터럽트 가능 설정
    EICRB = 0x0F;       // INT4,5 = 상승엣지 사용
    EIMSK |= 0x30;      // INT4,5 interrupt enable
    sei();

    // 동기화 개체 생성

    FndValSem = OSSemCreate(1);         // 전역변수 FndValue와 FndDigit변수를 보호하기 위한 세마포어
    ToneSem = OSSemCreate(1);           // Tone변수 중복 접근을 막기위한 세마포어
    e_grp = OSFlagCreate(0x00, &err);   // TIMER1 인터럽트 서비스 루틴이 인터럽트가 발생했음을 알리는 프래그
    TempMbox = OSMboxCreate((void *)0); // 온도측정하는TempTask가 온도가 한도치를 초과했는지를 메시지로 LedTask에게 보냄

    // 디바이스 상태값 초기화
    timer_state = OFF;          // 타이머 상태 = 꺼짐
    buzzer_state = OFF;         // 부저 상태 = 꺼짐
    mute = 0;                   // 쉼표, 무음모드 = 아님

    // 1개의 시간타이머 태스크와 3개의 디바이스 제어 태스크 생성
    OSTaskCreate(WatchTask, (void *)0, (void *)&WatchTaskStk[TASK_STK_SIZE - 1], WATCH_PRIO);
    OSTaskCreate(LedTask, (void *)0, (void *)&LedTaskStk[TASK_STK_SIZE - 1], LED_PRIO);
    OSTaskCreate(DispFndTask, (void *)0, (void *)&DispFndTaskStk[TASK_STK_SIZE - 1], DISP_PRIO);
    OSTaskCreate(TempTask, (void *)0, (void *)&TempTaskStk[TASK_STK_SIZE - 1], TEMP_PRIO);

    OSStart();      //멀티태스킹 시작

    return 0;
}
```

3) WatchTask : 오븐 타이머 처리 task

사용자가 입력한 시간에서 시간을 줄여나가는 카운터 기능을 하는 태스크

① TIMER1을 값비교 인터럽트로 설정한다. 분과 초사이의 점을 깜박이게 하기 위하여 0.5초 단위로 인터럽트를 발생시킨다.

② resetTimer를 호출하여 타이머를 초기화한다.

③ 무한 루프를 돌면서 다음을 반복한다.

I. 타이머 ISR로 부터 이벤트그룹의 플래그가 세팅되길 기다린다. 0.5초가 흐른것임

II. 짝수번째이면, FndDigit = 0x00(점 안찍음), 홀수번째이면, FndDigit = 0x04(우측셋째자리에 점찍음)을 반복 출력하여, 1초 단위로 점이 깜박이게 만든다.

III. 짝수 번째마다 seconds 를 1씩 줄이고 0이되면 buzzerOn함수를 호출하여 알람음악을 울리고, 타이머를 0으로 리셋한다.인터럽트도 disable한다.

IV. 현재의 seconds값을 FND에 표출할 수 있도록 FndValue에 변환하여 출력한다. 변환 공식은 seconds/60 * 100 + seconds%60 이다.

```
void WatchTask(void *pdata)
{
    INT8U   err;
    unsigned int halfseconds = 0;

    pdata = pdata;

    OS_ENTER_CRITICAL();
    TCCR1B = (1 << CS12 | 1<<CS10 | 1 << WGM12);
    //OCR1A = 15625 - 1;            //1초
    OCR1A = 7813 - 1;       //0.5초
    TIMSK |= 1 << OCIE1A;
    OS_ENTER_CRITICAL();

    resetTimer();           //오븐의 타이머를 초기화한다.

    for (;;){
        //Flag가 셋팅될 때까지 기다린다.이 값은 TIMER1 ISR에서 0.5초마다 세팅한다.
        OSFlagPend(e_grp, 0x01, OS_FLAG_WAIT_SET_ANY + OS_FLAG_CONSUME, 0, &err);

        halfseconds++;
        halfseconds %= 2;

        if (!halfseconds)
        {
            OSSemPend(FndValSem, 0, &err);  // DispFndTask랑 공유하므로 보호
            FndDigit = 0x00;                // 소숫점을 표시하지 않음
            OSSemPost(FndValSem);

            if ((--seconds) == 0)       //설정된 시간에 다다름.
            {
                buzzerOn();             // 알람 태스크를 생성하고 음악을 재생한다.
                resetTimer();           // 오븐의 타이머를 초기화한다
            }

        }
        else
        {
            OSSemPend(FndValSem, 0, &err);
            FndDigit = 0x04;                // 소숫점 표시. 1초 단위 깜박임.
            OSSemPost(FndValSem);
        }

        OSSemPend(FndValSem, 0, &err);
        FndValue = seconds/60 * 100 + seconds%60;   // 초를 분.초로 변환하여 값을 설정
        OSSemPost(FndValSem);

        OSTimeDlyHMSM(0, 0, 0, 10);
    }
```

4) BuzzerTask : 알람 task

악보를 읽어서 TIMER2가 주어진 계명의 주파수 만큼 파형을 만들 수 있게한다. 악보는 계명과 박자의 2차배열로 선언하였다. 전역변수 tone에 해당 계명을 저장하고, OSTimeDlyHMSM 함수를 통해 박자수 * 120 ms 만큼 지연시킨다.

```
void BuzzerTask(void *pdata)
{

    /*
    int song[29][2] = {{SOL,4}, {MI,2}, {MI,2}, {SOL,2}, {MI,2}, {DO,2},{PS,2},
        {RE,4}, {MI,2}, {RE,2}, {DO,2}, {MI,2}, {SOL,2},{PS,2},
        {DDO,3}, {SOL,1}, {DDO,2}, {SOL,2}, {DDO,2},{SOL,2}, {MI,2},{PS, 2},
        {SOL,4}, {RE,3}, {FA,1}, {MI,2}, {RE,2}, {DO,4}, {EOS, 4}};*/
    int i,j;
    INT8U err;

    i=0;

    OS_ENTER_CRITICAL();
    DDRB = 0x10;        // 부저 출력방향 설정
    TCCR2 = 0x03;       // 8분주
    TCNT2 = f_table[song[i][0]]; // 타이머 초기값으로 첫번째 계명에 해당하는 카운터값 입력
    TIMSK |= 0x40;      // Overflow 인터럽트로 설정
    OS_EXIT_CRITICAL();

    for (j=0;j<2 ;j++ )     // 한곡을 2번 재생
    {

        i =0 ;
        OSSemPend(ToneSem, 0, &err);
        tone = song[i][0];              //첫번째 계명을 읽음
        OSSemPost(ToneSem);

        while (tone != EOS)             //계명이 악보의 끝이 아닐때까지 반복
        {

            mute = (tone == PS) ?  1 : 0;  //계명이 쉼표이면 전역변수 mute를 1로 설정 TIMER2 ISR에서 포트 출력을 막는다.
            OSTimeDlyHMSM(0,0,0,120*song[i][1]); //박자수 * 120 ms만큼 지연
            mute = 1;
            OSTimeDlyHMSM(0,0,0,10);        //음이 이어지지 않도록 잠시 끊어줌
            mute = 0;
            i++;                            // 다음계명을 읽어와서
            OSSemPend(ToneSem, 0, &err);
            tone = song[i][0];              // ISR이 읽어 갈수 있도록 tone 변수에 세팅
            OSSemPost(ToneSem);
        }
    }

    buzzerOff();        //2번 재생이 완료되면 태스크를 종료시키고, 인터럽트를 disable한다.

}
```

5) LedTask : 가열판 출력 task

① INT4에 의해 가동 혹은 중지임이 설정된 timer_state를 읽어 ON일 경우 가열판(LED)를 작동시킨다.

② 과열방지를 위해 과열정보를 TempMox에서 OSMboxAccept(TempMbox)함수로 읽어온다. TempTask는 0.5초 마다 온도센서에서 온도값을 읽어 임계치 보다 큰지의 여부를 OSMboxPost를 통해 전송해온다.

③ 과열이 아니거나 타이머가 ON중이면 LED 움직이는 패턴을 출력한다.

```
void LedTask (void *pdata)
{
    unsigned char direction = 0xff;
    unsigned char bit = 0x07;
    int bimetal, *msg;

    pdata = pdata;

    bimetal = ON;                       // 과열 아님
    DDRA = 0xff;                        // LED PORTA 출력으로 설정

    for(;;)
    {
        msg = (int *)OSMboxAccept(TempMbox);    //TempMbox에서 wait 없이 읽어옴. LED의 원활한 출력을 위해
        if (msg != 0)
        {
            bimetal = *msg;             //과열 정보 수신
        }

        // LED display

        if ((timer_state == ON) && (bimetal == ON)) //과열이 아니면서 타이머가 ON중이면 LED 움직이는 패턴을 출력한다.
        {
            PORTA = bit;
            if(bit == 0x07 || bit == 0xE0)          // 3비트씩 연속으로 좌우로 움직이는 패턴 표시
                direction = ~direction;
            bit = direction? (bit >> 1) : (bit << 1);
        }
        else
            PORTA = 0;                              // 꺼졌거나 과열이면 가열판을 끈다.

        OSTimeDlyHMSM(0,0,0,100);
    }
}
```

6) TempTask : 온도센서에서 온도를 측정하고 과열 여부를 판단하는 task

① I2C 통신을 통해 온도센서에서 온도값을 읽어와 실수로 변환한다.

② 시연의 용이성을 현재의 온도 + 0.5도를 threshold(임계값)으로 사용한다.

③ 무한루프를 돌면서 ATS75로 부터 온도 값을 읽어 임계값을 지나쳤는지 확인한다.

④ Tempbox 메시지박스에 포스팅하여, LEDTask가 과열 여부를 알 수 있도록 한다.

```
void TempTask(void *pdata) {
    int value;
    float temperature, threshold ;
    unsigned short value_int, value_deci;
    int bimetal;

    init_twi_port(); // TWI 및 포트 초기화

    write_twi_1byte_nopreset(ATS75_CONFIG_REG, 0x00); // 9비트, Normal
    OSTimeDlyHMSM(0, 0, 0,100); // 다음 사이클을 위하여 잠시 기다림

    value = read_twi_2byte_nopreset(ATS75_TEMP_REG);
    value_int = (char)((value & 0x7f00) >> 8);
    value_deci = (char)(value & 0x00ff);
    temperature = value_int + ((value_deci & 0x80) == 0x80) * 0.5;  //측정값을 실수로 변환
    threshold = temperature + 0.5;                      //현재값 + 0.5를 임계치로, 0.5도만 상승해도 꺼지도록
    OSTimeDlyHMSM(0, 0, 0, 500);                        // 0.5초지연
```

```
    while (1) {         // 무한 루프를 돌면서 온도값을 읽어 과열여부를 판단후 Mbox에 포스팅한다.
        value = read_twi_2byte_nopreset(ATS75_TEMP_REG);
        value_int = (char)((value & 0x7f00) >> 8);
        value_deci = (char)(value & 0x00ff);

        temperature = value_int + ((value_deci & 0x80) == 0x80) * 0.5;

        bimetal = (temperature > threshold) ? OFF : ON;
        //FndValue = value_int;

        OSMboxPost(TempMbox,(void*)&bimetal);
        OSTimeDlyHMSM(0, 0, 0,500);
    }
}
```

7) DispFndTask : fnd에 숫자를 출력하는 task

① 태스크 시작과 함께 PORTC와 PORTG의 출력방향을 아웃풋으로 설정한다.

② 세마포어 획득 후 WatchTask와 ISR5가 설정한 FndValue와 FndDIgit을 읽어와 FND에 출력한다

③ FndDigit bit의 값에 따라 소숫점을 찍는다.

```
void DispFndTask(void *pdata)
{
    int i;
    unsigned short num[4];
    int value;
    unsigned char point;
    INT8U   err;

    DDRC = 0xff;        // PORTC 출력
    DDRG = 0x0f;        // PROTG 출력

    while (1)
    {
        OSSemPend(FndValSem, 0, &err);  //세마포어 획득
        value = FndValue;               //전역변수를 읽음
        point = FndDigit;
        OSSemPost(FndValSem);

        num[3] = (value / 1000) % 10;   // 각 자리수 구함
        num[2] = (value / 100) % 10;
        num[1] = (value / 10) % 10;
        num[0] = value % 10;

        // 이자리에 출력
        for (i = 0; i < 4; i++) {
            PORTC = digit[num[i]];      //해당 숫자에 해당하는 비트열 출력
            PORTG = fnd_sel[i];
            if (point & BIT(i))         //FndDigit의 해당 자리 비트가 세팅되어 있으면 점 출력
                PORTC |= 0x80;
            OSTimeDlyHMSM(0, 0, 0,2);
        }
        //OSTimeDlyHMSM(0, 0, 0, 50);
    }
}
```

8) ISR(TIMER1_COMPA_vect)

16비트 timer 인 TIMER1에 대한 인터럽트 서비스 루틴이다. TIMER1의 값 비교 인터럽트를 사용하며, 0.5초에 1회 발생하도록 값을 설정하였다.

```
OS_ENTER_CRITICAL();
TCCR1B = (1 << CS12 | 1<<CS10 | 1 << WGM12);
//OCR1A = 15625 - 1;            //1초
OCR1A = 7813 - 1;       //0.5초
TIMSK |= 1 << OCIE1A;
OS_ENTER_CRITICAL();
```

인터럽트가 발생할 경우, 이벤트 플래그 e_grp에 1비트를 세팅한 후 포스팅하여 watch task에 0.5초가 지났음을 알린다.

```
ISR(TIMER1_COMPA_vect)
{
    INT8U    err;
    OSFlagPost(e_grp, 0x01, OS_FLAG_SET, &err);
}
```

9) ISR(TIMER2_OVF_vect)

TIMER2에 대한 인터럽트 서비스 루틴이다. TIMER2의 오버플로우 인터럽트를 사용한다.

① 전역변수 mute 값이 1로 설정된 경우 무음(쉼표)이므로 바로 리턴한다.

② state 값을 비교하여 PORTB의 5번 비트를 셋하거나 언셋한다

③ state 값을 반대로 바꾸어 다음에 들어올 때 PORTB의 5번 비트 출력이 바뀔 수 있도록 한다.

```
ISR(TIMER2_OVF_vect)
{
    //INT8U err;

    if (mute)
        return;

    if (state == ON)
    {
        PORTB = 0x00;
        state = OFF;
    }
    else
    {
        PORTB = 0x10;
        state = ON;
    }
    //OSSemPend(ToneSem, 0, &err);
    TCNT2 = f_table[tone];
    //OSSemPost(ToneSem);

}
```

10) ISR(INT5_vect)

2번 단추 입력에 대한 인터럽트 서비스 루틴이다. Beep() 함수를 호출하여 짧은 버튼 음을 출력한 후,

① 부저가 켜져 있는 상태이면 부저를 끈다.

② 그렇지 않은 경우는 타이머를 세팅하는 기능을 한다. 즉 타이머 세팅값을 10초씩 증가시킨다.

```
ISR(INT5_vect)
{
    beep();

    if (buzzer_state == ON)
    {
        buzzerOff();
    }
    else
    {
        seconds += 10;
        FndValue = seconds / 60 * 100 + seconds % 60;
    }
    _delay_ms(10); // debouncing

}
```

11) ISR(INT4_vect)

1번 버튼 입력에 대한 인터럽트 처리 함수이다. 1번 버튼은 타이머가 작동 시에는 타이머를 일시정지 시키고, 다시 누르면 타이머가 다시 작동하게 한다. 즉, 타이머가 0이 아닐 때 인터럽트가 발생하면 타이머를 재시작 한다.

```
ISR(INT4_vect)
{

    beep();

    if (timer_state == ON)
    {
        pauseTimer();
    }
    else if (seconds > 0)
    {
        startTimer();
    }

    _delay_ms(10); // debouncing

}
```

12) pauseTimer(), resetTimer(), startTimer()

타이머가 작동 중인지를 나타내는 timer_state 전역 변수의 상태를 각각 PAUSE, OFF,

ON으로 설정하고 timer 마스크인 TIMSK의 비트 값을 설정한다. 즉, PAUSE 와 RESET 은 TIMSK 를 unset 하고, START는 set 한다.

```
void pauseTimer( )
{
    timer_state = PAUSE;
    TIMSK &= ~(1 << OCIE1A);
}

void resetTimer( )
{
    timer_state = OFF;
    seconds = 0;
    TIMSK &= ~(1 << OCIE1A);
}
void startTimer( )
{
    timer_state = ON;
    TIMSK |= 1 << OCIE1A;
}
```

13) buzzerOn(), buzzerOff()

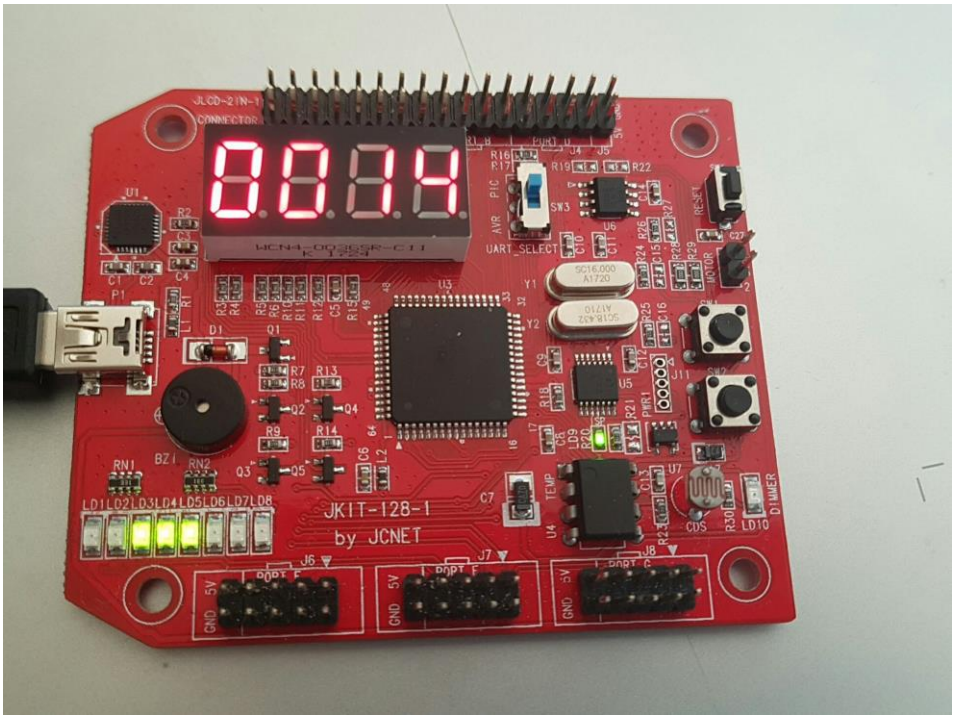
알람 곡 재생을 위해 부저를 켜고, 재생 중인 부저를 끄는 함수들이다. 다른 task들은 프로그램 시작 시부터 종료 시까지 무한 루프를 돌며 끊임없이 돌도록 설계하였으나 buzzer task 들은 부저가 울릴 필요가 있을 때에만 OSTaskCreate 하여 음악을 재생하고 task를 삭제시키는 방법을 사용하였다.

그래서 buzzerOn() 에는 버저 task를 생성하고, TIMSK를 0x40으로 oring 하여 TIMER2_OVF 인터럽트를 enable 시킨다.

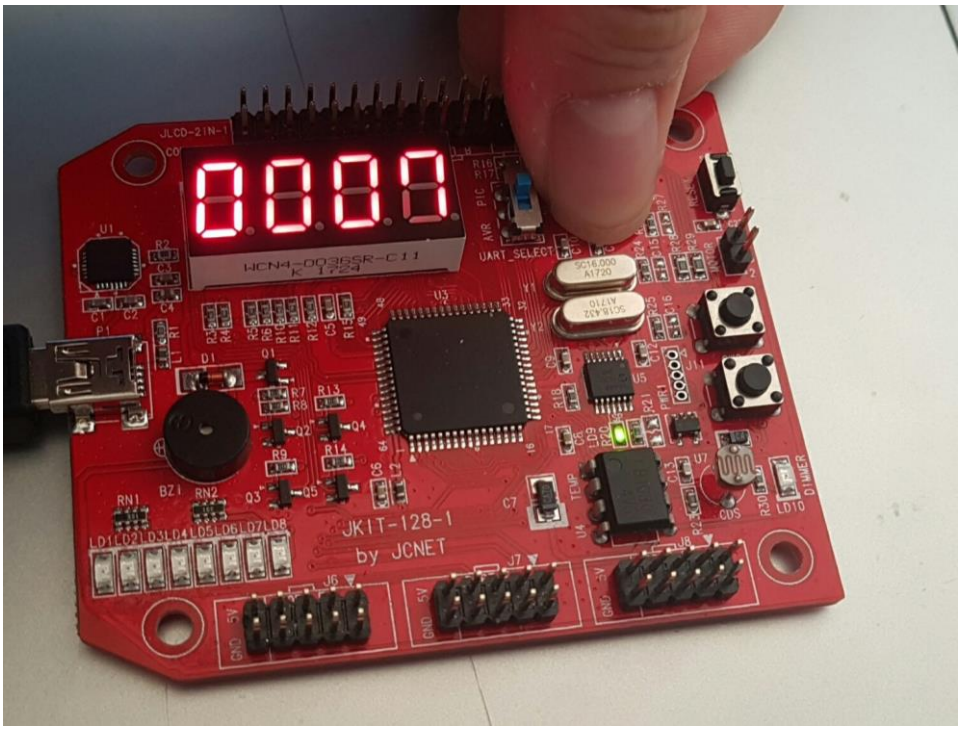
BuzzerOff()의 경우 TIMSK의 6번 비트를 unset 하고, OSTaslDel()로 종료하고 삭제시킨다.

```
void buzzerOn( )
{
    buzzer_state = ON;
    OSTaskCreate(BuzzerTask, (void *)0, (void *)&BuzzerTaskStk[BUZZERTASK_STK_SIZE - 1], 1);

    TIMSK |= 0x40; // Overflow
}

void buzzerOff( )
{
    buzzer_state = OFF;
    TIMSK &= ~(0x40); // Overflow
    OSTaskDel(BUZZ_PRIO);
}
```

6. 실행 화면



일반 실행 화면



0.5도 이상 온도를 증가시켰을 때 가열판이 멈추는 모습

7. 구현 시 어려웠던 점

- 부저 task만 실행 시키면 프로그램이 이상해지고 데이터가 깨지는 문제가 발생하였다. 전체적인 문제가 발생하였으나 디버깅이 어려워 문제점을 발견하는 데 오랜 시간이 걸렸다. 원인은 악보의 배열을 전역 변수가 아니라 buzzer task 의 지역변수로 선언하였는데, 전체 지역변수의 크기가 기본 task 스택 사이즈보다 커져서 스택이 깨져서 발생하는 문제였다. 그래서 Buzzer task 에는 기본 사이즈보다 4배 큰 스택 사이즈를 주어서 문제를 해결하였다. 또는 악보를 전역변수로 옮기면서 해결할 수 있었을 것으로 예상된다.

8. 결론

이 프로젝트를 통해 그동안 배웠던 uC/OSII 의 멀티 테스킹의 기본 개념을 더 깊이 이해할 수 있었고 동기화 기법의 필요성을 깨달았으며, 또한 인터럽트, 타이머, 디바이스 입출력, I2C 통신 등 임베디드 시스템의 다양한 디바이스들과 통신하는 다양한 기법에 대해 충분히 숙지할 수 있었다.