임베디드 소프트웨어 경진대회
The World Embedded Software Contest

제16회 임베디드SW경진대회

# 개발완료보고서 [실시간 IoT시스템]

산업통상자원부

KESSIA 임베디드소프트웨어·시스템산업협회
Korea Embedded Software and System Industry Association

## 1. 작성 시 주의사항

**※ 제출 분량 : A4 용지 상세내용 포함 20 page 이내 ( 최대 폰트 크기 12pt )**
**※ 제출 포맷 : pdf**

## 2. 팀 정보

| 팀명 | MSDE | 팀장 | 한승한 |
|---|---|---|---|
| 팀원 | 박형주 | 팀원 | 최용래 |
| 팀원 | 황형준 | 팀원 | 양재필 |

## 3. 본 개발완료보고서

### 0. RTOS 기반 음파소화기 시스템

#### 1. 개요

##### 1.1. 작품 개요

본 아이디어는 음파소화기를 이용하여 스토브와 같은 난방 및 전열기에서 화재가 났을 시, 이를 화재감지센서를 이용하여 감지하고, 음파증폭기와 스피커로 일정주파수와 데시벨 음파를 이용해 화재 초기 진압을 하는 기능을 가지고 있다. 화재가 난 제품의 상황을 실시간으로 확인할 수 있으며, 에너지 사용의 효율성을 높이기 위한 실시간 IoT시스템 화재진압 안전 스토브에 대한 설명이다.

본 실시간 시스템을 사용하기 위해 한컴MDS사의 NEOS(RTOS)를 사용하였으며, 제안하는 음파 소화 시스템은 RTOS의 다음과 같은 특징으로 시스템에 가장 적합한 OS이므로 선정하였다.

첫째, Reactive한 시스템에 좋다.

둘째, Asynchronous 반응에 즉각 대응한다.

셋째, Time constraints를 충족하여 Hard-Realtime-Scheduling이 가능하게 한다.

넷째, Reliable 하다. Thread가 정해진 시간 안에 작업을 무조건 완료한다.

다섯째, Predictable 하다. 작업이 분할하여 구성하기 때문에 복잡한 시스템이더라도 다음 동작을 쉽게 예측할 수 있다.

본 시스템은 사용 횟수에 제한적이지 않으며, 화재를 초기에 진압하여 2차 피해에 대한 조기 대처가 가능하고, 화재 시 화재 발생 스토브의 전기 또는 가스 공급기를 자동으로 차단하여 화재의 규모를 한정시키며, 사람이 없는 상황에서 발생한 화재에 대한 대처가 가능하도록 화재 발생 유무에 따른 신속한 대처를 기대할 수 있다. 본 소화 시스템은 일반 소화기와 다르게 분사물이 없어 화재 이후 2차 피해가 없는 공명기를 이용한다. 이 시스템의 화재 진압기능은 가전제품에 접목하여 화재 발생 시 화재 대처가 초기에 가능하도록 시스템을 구성시킬 수 있다. 또한, 더 나아가 화재 감지 및 화재진압 동작을 위한 전기에 너지 소모를 최소화하기 위한 차단기를 구성하여 실제 제품에도 충분히 도입 가능하도록 제작하였다.

##### 1.2. 필요성

일반 소화기의 사용에는 일회성 소비에 한정적이며, 이에 따른 경제적 부담이 생긴다. 하지만, 음파소화기 시스템은 사용 횟수에 제한적이지 않으며, 화재를 초기 진압하여 2차 피해에 대한 조기 대처가 가능하다. 화재 대처 능력에서 일반 가정이나 음식점에서 조리기구 사용 중 발생한 화재 등에 단순히 스프링클러를 통한 화재 진압과 작업자가 직접 일반 소화기를 찾은 후 화재 진압을 작업자가 수행해야하는 점에 비해 화재 감지를 통한 실시간 대처가 가능하고, 초기에 대처를 하기 때문에 인명피해나 경제적인 피해를 최소화시킨다. 화재 시, 화재 발생 스토브의 전기 또는 가스 공급기를 자동으로 차단하여 화재의 규모를 한정시켜 별도의 분사 없이 화재 진압이 동작하기 때문에, 비교적 깔끔한 대처가 가능하다.

제안하는 본 시스템은 화재진압기능을 가진 가전제품 장치 안에 음파 소화기를 장착하는 것을 목표로 하며, 분리하여 탈착이 가능하도록 모듈을 구성하는 것이 가능하도록 장비의 유동성을 높이게 구성하였으며, 사용자의 편의성을 높이도록 방안을 모색하였다. 결과적으로, 사람의 물리적인 행위 없이 화재 진압에 대한 초동 대처가 가능하도록 시스템을 구성하고자 한다.

##### 1.3. 개발 목표

개발 목표는 5가지의 단계로 구성하였다.

첫째, 음파소화기의 효율성을 높이기 위해 최적화된 상태의 공명주파수와 데시벨을 구하고, 일반적인 경우에 음파소화기가 작동이 가능하도록 여러 경우의 화재에 대한 연구를 진행하였다.

둘째, 가스레인지, 전기오븐 등으로 이루어진 가정용 조리기구 일면에 위치되어, 연기 및 불꽃을 감지하는 화재감지 센서와 센서데이터를 처리하는 마이크로컨트롤러, 음파증폭기, 전원장치 H/W를 설계한다.

셋째, 실시간으로 화재감지 센서의 센싱 신호를 입력받아 기준 설정 치와 비교분석 후 화재유무가 판별되면, 실시간으로 음파 소화기의 전원스위치를 ON 상태로 하여 화재 진압을 작동시키고 안정적인 화재진압이 이루어지도록 설계하고 구현한다.

넷째, 화재 감지 및 작동에 전기에너지 사용량을 최대한 줄이고자, 차단기를 사용하여 구성한다.

마지막으로, 실시간 IoT 시스템 부문에서 지원하는 한컴MDS사의 NEOS(RTOS)를 활용하여 동작 Thread들의 Hard-Realtime-Scheduling을 통해 음파소화기를 제어한다.

## 2. 작품 설명 (최대한 자세하게 기술)

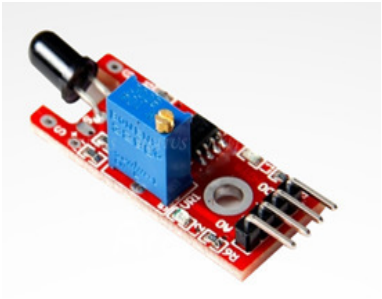
### 2.1. HW 구성

#### 2.1.1. 스피커: 0610-4044FB (MARSHALL SOUND)

| Specification | value |
|---|---|
| Power Handling(RMS) | 100 [W] |
| Power Handling(max) | 200 [W] |
| **Impedance** | **4 [ohms]** |
| **Frequency Response** | **45~3500 [Hz]** |
| Sensitivity | 89 [dB]±2% |
| Voice Coil Diameter | ±0.1 [mm] |
| Outside Nominal Diameter | 170 [mm] |
| Total Height | 80 [mm] |
| **Weight** | **0.8 [kg]** |

[제품 선정 근거]
1. 가벼운 무게
2. 작은 파워
3. 넓은 범위의 주파수 반응

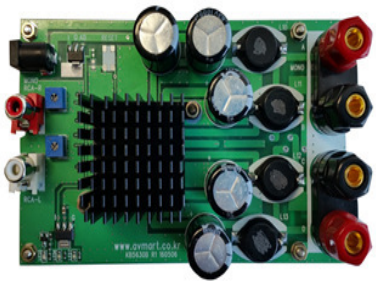
#### 2.1.2. 불꽃 감지 적외선 센서 IR 모듈 (AS0358)



| Specification | value |
|---|---|
| Detect wave length range | 760 ~ 1100 [nm] |
| **Detect distance range** | **50 ~ 80 [cm]** |
| Operating Voltage | 3.3 ~ 5 [V] |
| Operating Current | 10 [mA] |
| Size | 32 X 14 [mm] |

[제품 선정 근거]
1. 화재 감지의 반응성이 가장 적합한 모듈 선택
2. 저렴한 가격대

### 2.1.3. Amplifier Board: 5630 ST (TI)



| Specification | value |
|---|---|
| Channels | 2 Channels |
| **Output Power** | **300 [W]** |
| **Impedance** | **2 ~ 8 [ohms]** |
| Operating Voltage | 18 ~ 50 [V] |
| Input Interface | mono signal |
| PCB size | 133 x 83 x 33 [mm] |

[제품 선정 근거]
1. 우리가 사용하는 스피커 스펙과 적합한 증폭기 선택

### 2.1.4. Collimator

| Specification | value |
|---|---|
| Material | Melamine |
| Outside Diameter | 170 [mm] |
| Inside Diameter | 160 [mm] |

[제품 선정 근거]

1. 화재 발생 시, 물리적인 힘과 열을 버틸 수 있는 내열성, 고경도 소재인 멜라민 선택

### 2.1.5. Servo motor (HS-311 LSIS)


| Specification | value |
|---|---|
| Rated Voltage | 4.8 ~ 6 [V] |
| Rated Frequency | 50/60 [Hz] |
| Size | 41 x 20 x 37 [mm] |
| No-load Speed | 0.15 sec/60° |

[제품 선정 근거]
1. 화재 감지 시, 전기 및 가스 공급 밸브를 차단하기 적절한 서보모터 선택

### 2.1.6. BK63H Circuit Breaker (LSIS)


| Specification | value |
|---|---|
| Rated Voltage | DC 60 ~ 220 [V]<br>AC 110 ~ 380 [V] |
| Rated Frequency | 50/60 [Hz] |
| Size | 18 x 105 x 66 [mm] |
| Short Circuit Capacity | 10 [kA]<br>@ 120/240 AC, 1 ~ 63 A |

[제품 선정 근거]
1. 회로의 과부하와 단락을 방지하기 위해 선정

### 2.1.7. Relay Board


| Specification | value |
|---|---|
| Operating Relay Voltage | DC 5 [V] |
| Max AC | 250[V] / 10[A] |
| Max DC | 30[V] / 10[A] |
| Size | 54 x 31 x 18.5 [mm] |

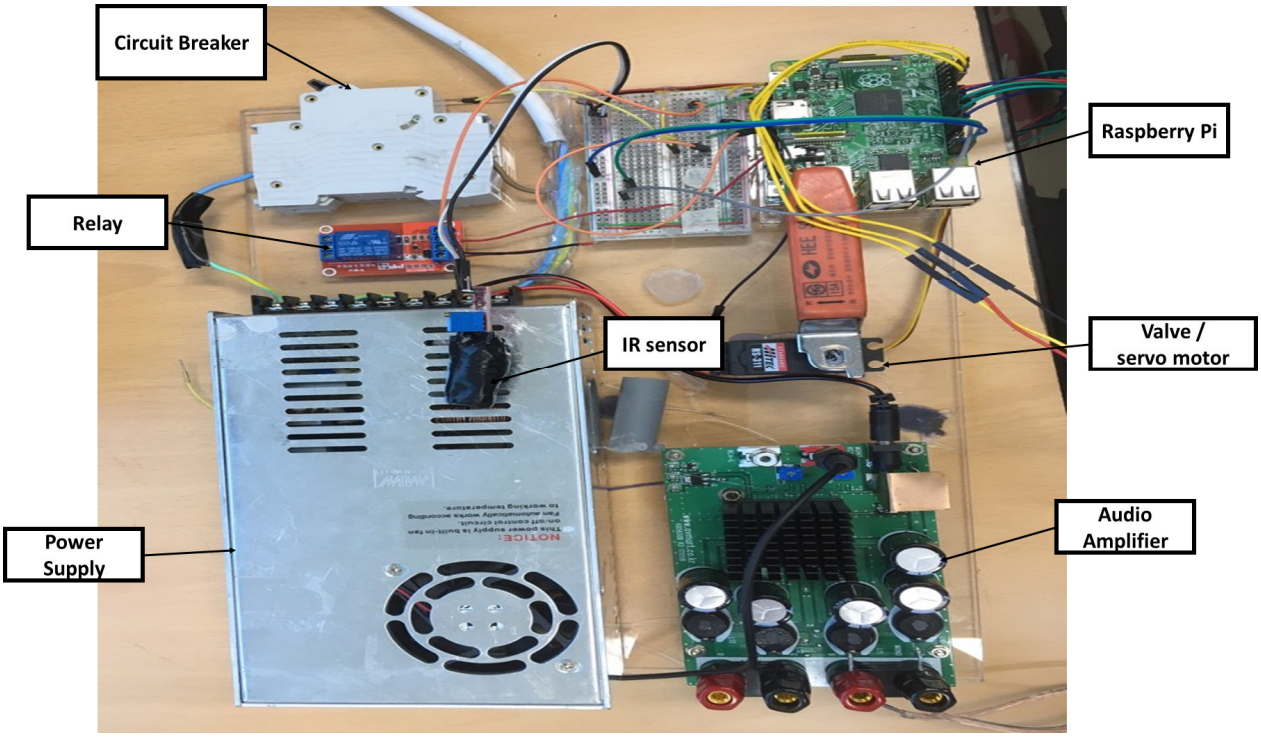
[제품 선정 근거]
1. 회로 내에서 스위칭 작용을 하기 위해 선정

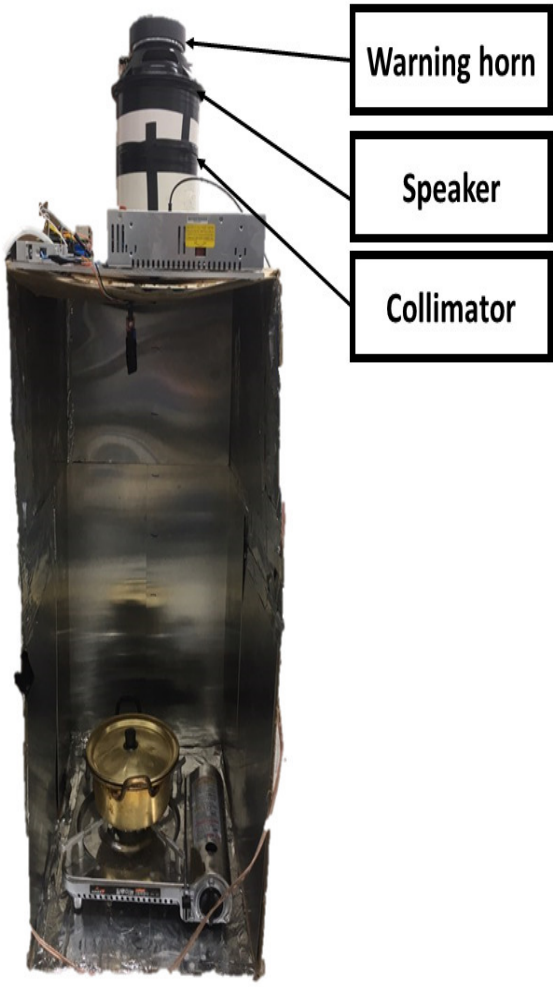
### 2.1.8. Power Supply: NES-350-48 (Mena Well)


| Specification | value |
|---|---|
| Output DC Voltage | 48 [V] |
| Output Rated Current | 7.3 [A] |
| Output Current range | 0 ~ 7.3 [A] |
| Output Rated Power | 350.4 [W] |
| Input Voltage Range | AC 184 ~ 264 [V] |
| Size | 115 x 50 x 115 [mm] |

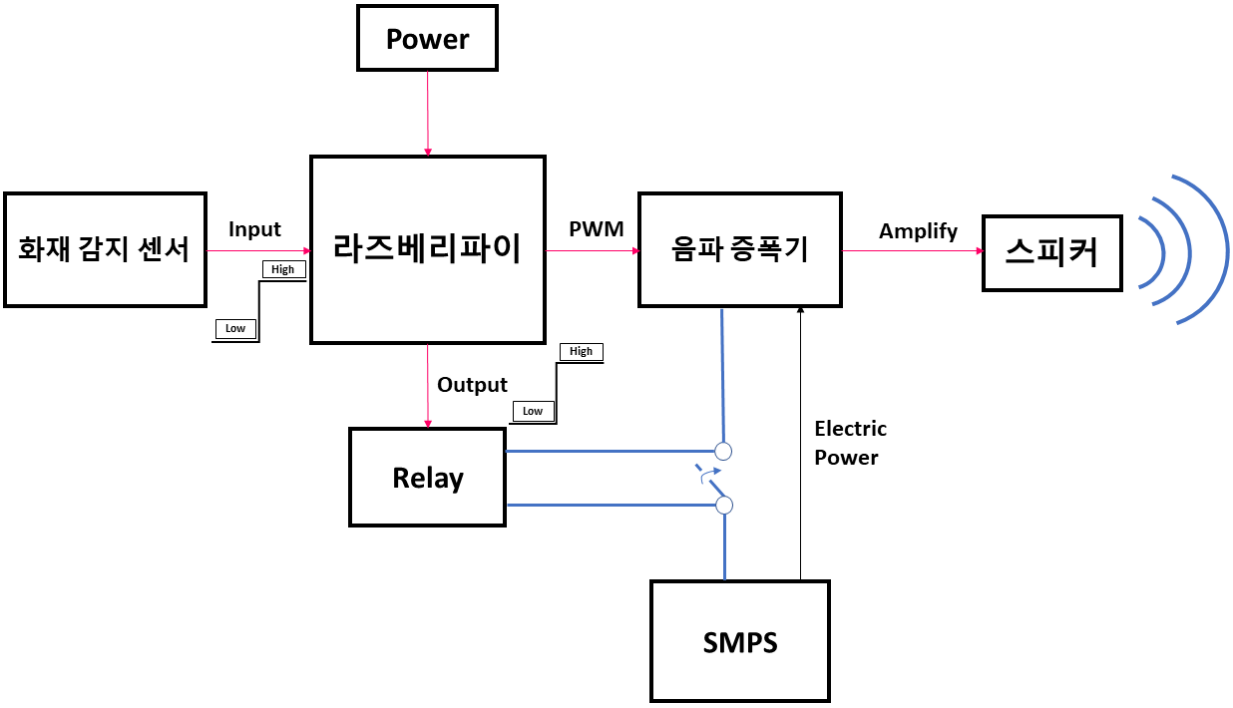
[제품 선정 근거]
1. 음파 증폭기에 원하는 전류와 전압을 공급하기 위해 선정

2.1.9. 회로 구성도 및 HW 전체 구성도



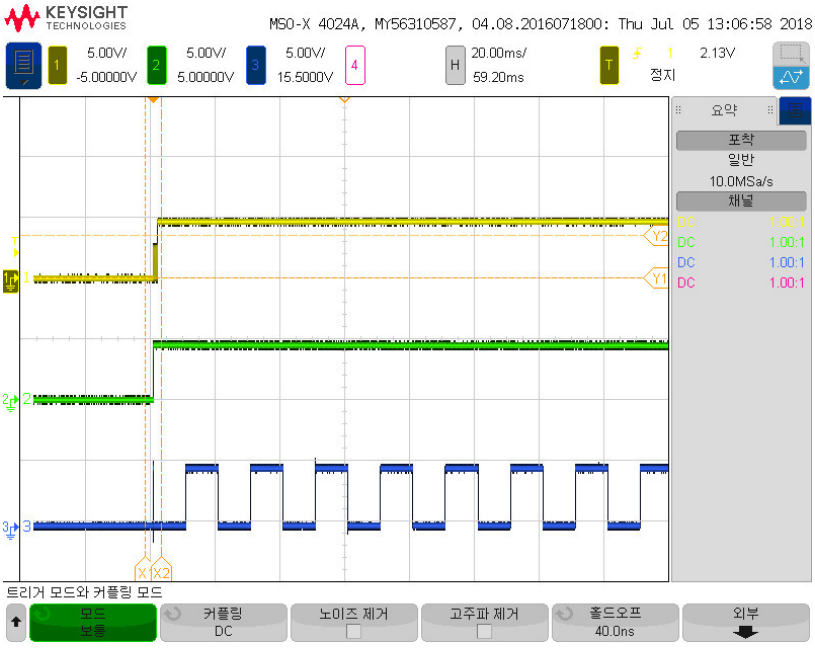




## 2.2. HW 기능(제어 방법 등 서술)

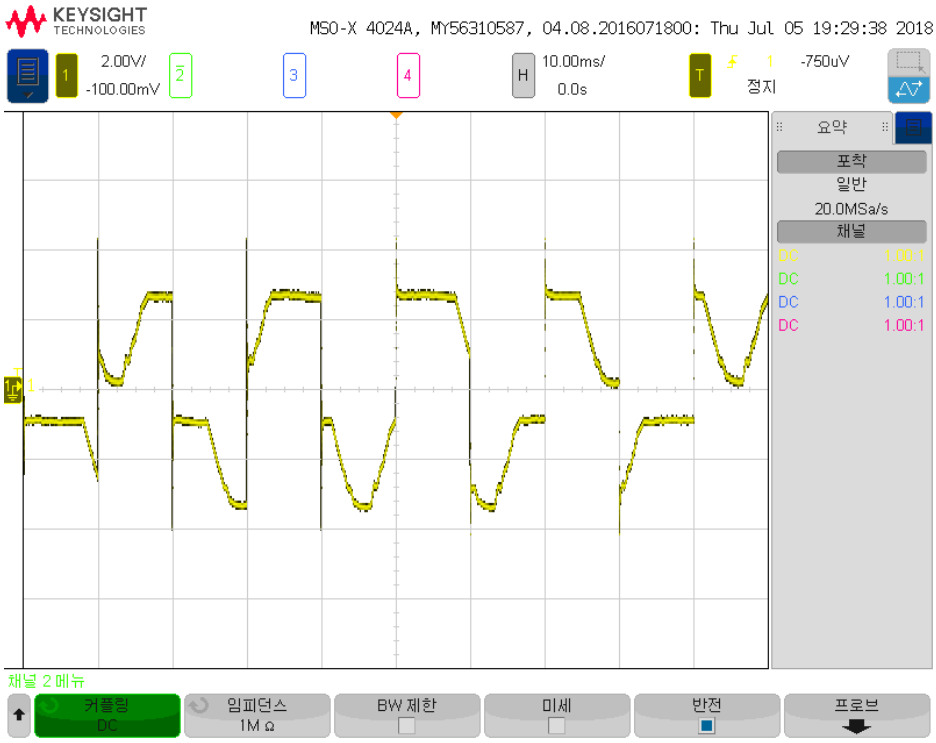



1. 마이크로컨트롤러인 라즈베리파이가 Power에 의해 전력 공급을 받는다.
2. 화재 감지 센서에서 연속적으로 감지 동작을 한다. 화재가 발생 시, 화재 감지 센서에서 High 신호를 라즈베리파이에 전달한다.
3. 화재 발생 신호를 전달받은 라즈베리파이는 Relay에 신호를 전달하여 개폐장치를 닫음으로써 음파 증폭기에 전력을 공급시킨다.
4. 라즈베리파이로부터 음파 증폭기는 신호를 받아 적절한 PWM 파형을 전달 받는다.
5. 음파 증폭기와 연결된 스피커에 신호를 전달하여 최종적으로 적절한 음파를 만든다.

실험을 통한 제어 파형을 오실로스코프를 이용하여 분석을 하면, 화재 감지 센서(CH1), Relay Output(CH2), PWM output(CH3)이 다음 그림과 같이 보여진다. 처음 화재 감지 센서가 HIGH(1)로 작동하자, Relay에 의해 전자개폐장치가 닫히고(HIGH,1), 음파증폭기가 스피커로 내보내는 PWM 파형이 출력되는 화면을 볼 수 있다. 이로써, 우리는 적절한 제어 장치를 통해 목표를 구현한 모습을 보였다.



음파 증폭기를 이용한 스피커 증폭 출력파형은 다음과 같다. 실제 화재 발생 상황을 가정한 실험에서 성공적으로 화재를 초기에 진압하는 모습을 보였으며, 안정적으로 작동하는 모습을 볼 수 있었다.




## 2.3. Software 구성

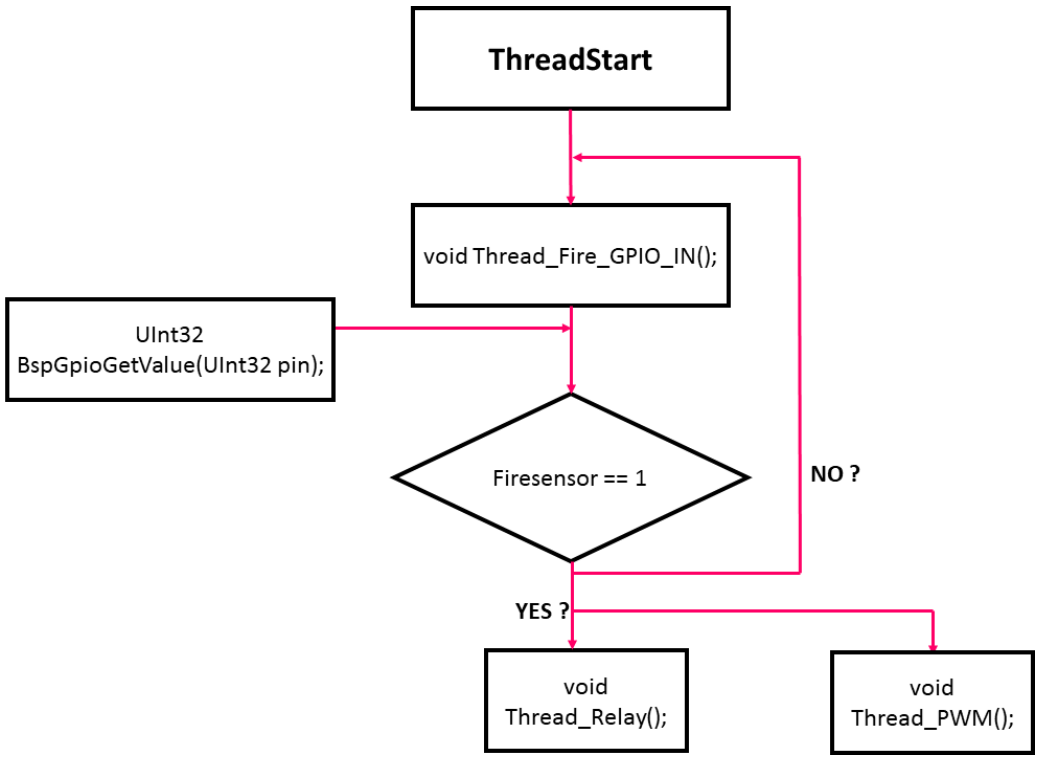
본 시스템은 실시간성을 요구하는 시스템으로써, 한컴MDS사의 NEOS를 사용하여 소프트웨어를 구성하였다. RTOS의 Multi-Threaded Process를 활용하여 총 3가지의 Tread를 구성하였고, 각 Tread마다 사용되는 함수와 우선순위를 설정하였다. Main 함수인 UserAppInit(void) 내에 3개의 Tread를 선언하였고, 각 Tread마다 함수는 다음 표와 같이 정리하였다.

| Thread - 1<br>(Fire detection sensor)<br><br>void Thread_Fire_GPIO_IN();<br>UInt32 BspGpioGetValue(UInt32 pin); | Thread - 2<br>(Relay)<br><br>void Thread_Relay(); | Thread - 3<br>(PWM)<br><br>void Thread_PWM(); |
|---|---|---|
| 우선순위 : 1 | 우선순위 : 2 | 우선순위 : 3 |

처음 Thread - 1에서 화재 감지 센서를 통해 화재유무를 판단하여 화재가 감지되지 않는다면, Thread - 2와 Thread - 3에서는 반응을 하지 않도록 설계했으며, 만약 화재가 센서를 통해 감지가 된다면, Thread - 1에서 firesensor 변수 값을 True(1)로 설정하여 Thread - 2와 Thread - 3에서 각각 전자개폐기를 닫고 PWM파형을 출력하도록 구성하였다.

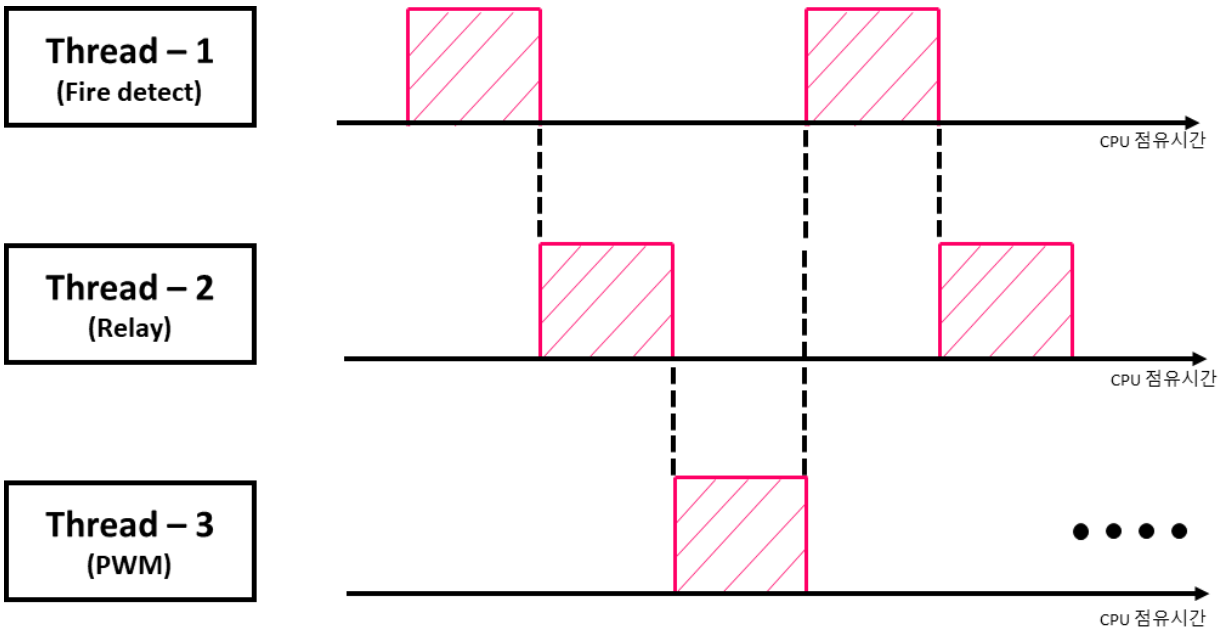
## 2.4. Software 설계도(흐름도 및 클래스 다이어그램 등 (개발언어에 따라 선택))

<Flow Chart>




본 시스템의 Software 설계도는 위 그림의 <Flow Chart>로 설명한다.

## 2.5. Software 기능 (알고리즘 설명 포함)




먼저, Software 구성에서 언급한 3가지의 Thread에 우선순위를 두어 Multi-Threaded Process를 진행하였다. 위 그림과 같이 한 주기 내에 우선순위대로 Thread가 동작하며, 실시간으로 Thread들이 병렬로 Task가 처리되어지고 있다. 2.4의 Software 설계도에서 언급되었듯이 각 Thread들 간의 상호 작용을 통해 음파 소화기 시스템의 최종 목표인 적절한 음파가 증폭되어 스피커를 통해 만들어지도록 구성하였다.

## 2.6. 프로그램 사용법 (Interface)

UBOOT에서 설정 방법

1. MAC 주소 설정

U-BOOT> setenv ethaddr 'b8:27:eb:c7:53:61'

2. IP 주소 설정

U-BOOT> setenv ipaddr '192.168.1.130'

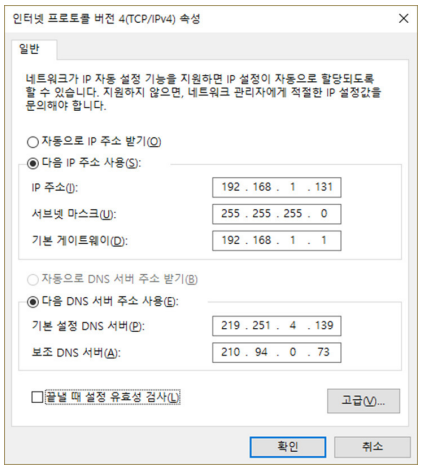
3. 서버 주소 설정

U-BOOT> setenv serverip '192.168.1.131'

4. 설정 내용 저장

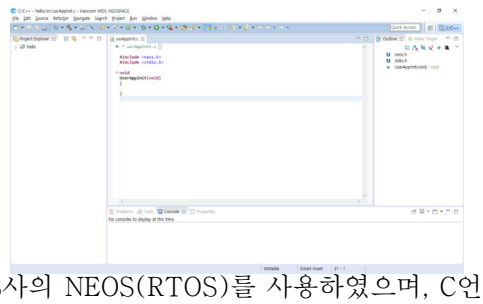
U-BOOT> saveenv

소프트웨어 환경 설정 - PC 네트워크 환경 설정 및 네트워크 동작 확인




- NEOSPACE-contest-arm을 통해 통합환경개발환경(NEOSACE)를 실행
- NEOS Project를 새로 만들고, 커널 이미지 프로젝트를 생성, 빌드
- 커널 이미지 다운로드를 위해 TFTP 서버 설정을 한 후 커널 이미지 다운로드 및 실행
- 설정 방법 -> setenv neos 'tftp 0x10000 192.168.1.131:ta.bin; go 0x10000'
  saveenv
- 배치 명령 실행 방법
  run neos
- github에 올라온 "src" 폴더의 usrAppInit.c 파일을 오픈
- 빌드 후 커널 이미지 생성, reboot 쉘 명령어를 이용하여 부트로더 상태로 재부팅
- 커널을 다시 다운로드하고 실행하여 소스 코드 실행

## 2.7. 개발환경 (언어, Tool, 사용시스템 등)



본 시스템은 한컴MDS사의 NEOS(RTOS)를 사용하였으며, C언어를 기반으로 Source code를 작성하였다. 마이크로컨트롤러는 라즈베리파이2 Model B V1.1을 사용하였다.

## 3. 프로그램 설명 (최대한 자세하게 기술)

### 3.1. 파일 구성

Source code는 usrAppInit.c에 구성되어 있으며, Release폴더 내에 ta.bin파일을 통해 커널 이미지를 제공한다.

### 3.2. 함수별 기능

#### 3.2.1. Thread_Fire_GPIO_IN();

이 함수는 화재감지센서로부터 마이크로컨트롤러에 신호를 받도록 하는 함수이다.

함수의 구성은 다음과 같다.

```
void Thread_Fire_GPIO_IN()
{
    // GPIO Input for firesensor

    BspGpioSetAlt(GPIO_PIN_NUM_06, GPIO_GPFSEL_IN);

    while(1)
    {
        firesensor = BspGpioGetValue(GPIO_PIN_NUM_06);
    }
}
```

BspGpioSetAlt는 GPIO 6번핀을 INPUT으로 받도록 한다.

BspGpioGetValue 함수는 다음과 같다.

```
UInt32 BspGpioGetValue(UInt32 pin)
{
    const UInt32 regId = (pin / GPIO_GPSET_PINS_PER_REG);
    const UInt32 shift = (pin % GPIO_GPSET_PINS_PER_REG);
    UInt32 value;
    UInt32 maskval;

    maskval = 1 << shift;

    value = REG32_READ(GPIO_PHY_ADDR, GPIO_GPLEV(regId));

    return ((value & maskval) >> shift);
}
```

BspGpioGetValue 함수는 UInt32 자료형의 pin을 입력받아 UInt32 자료형으로 출력한다.

GPIO 데이터 읽기 기능을 수행하는 BspGpioGetValue함수를 구성한다.

#### 3.2.2. Thread_Relay();

```
void Thread_Relay()
{
    // GPIO Output for Relay

    BspGpioSetAlt(GPIO_PIN_NUM_22, GPIO_GPFSEL_OUT);
    while(1)
    {
        if(firesensor == 1)
        {
            BspGpioSetValue(GPIO_PIN_NUM_22, 1);
            ThreadDelay(100000000ULL);
        }
        else
        {
            BspGpioSetValue(GPIO_PIN_NUM_22, 0);
            ThreadDelay(100000000ULL);
        }
    }
}
```

BspGpioSetAlt함수를 통해 22번 핀을 OUTPIN으로 설정한다.

firesensor 값이 1이라면, 22번 핀을 통해 출력을 True(1)로 설정하고, ThreadDelay를 한다.

firesensor 값이 0이라면, 22번 핀을 통해 출력을 False(0)으로 설정하고 ThreadDelay를 한다.

### 3.2.3. Thread_PWM();

```
void Thread_PWM( )
{
    // GPIO Output for PWM out

    BspGpioSetAlt(GPIO_PIN_NUM_21, GPIO_GPFSEL_OUT);
    while(1)
    {
        if(firesensor == 1)
        {
            BspGpioSetValue(GPIO_PIN_NUM_21, 0);
            //printf("LED1 OFF₩n");
            ThreadDelay(100000000ULL);
            BspGpioSetValue(GPIO_PIN_NUM_21, 1);
            //printf("LED1 ON₩n");
            ThreadDelay(100000000ULL);
        }
        else
        {
            BspGpioSetValue(GPIO_PIN_NUM_21, 0);
            ThreadDelay(100000000ULL);
        }
    }
}
```

BspGpioSetAlt함수를 통해 21번 핀을 OUTPIN으로 설정한다.
firesensor 값이 1이라면, GPIO 제어를 수행한다.
firesensor 값이 0이라면, 21번 핀을 통해 출력을 False(0)으로 설정한다.

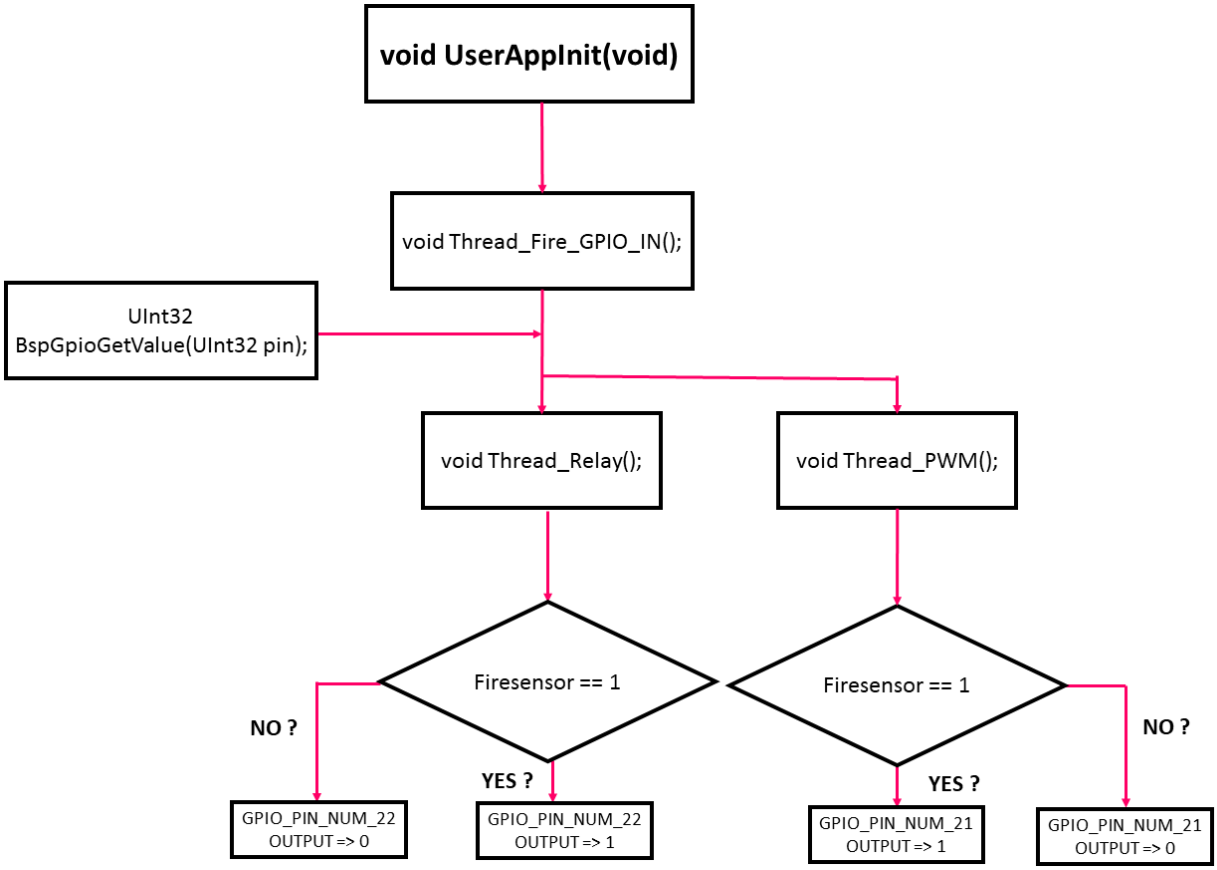
### 3.2.4. void UserAppInit(void);

```
void UserAppInit(void)
{
   printf("The sound wave extinguisher system starts operation₩n");

   Thread tid, tid2, tid3;
   int status;

   // PWM output thread for wave extinguisher system
   status = ThreadCreateEx("PWM", 4096, 100, (Address)Thread_PWM, NULL, &tid);
   if(status != SUCCESS)
   {
      printf("%s>%s thread creation failed₩n", __FUNCTION__, Thread_PWM);
   }
   else
   {
      printf("%s>%s thread creation succeed₩n", __FUNCTION__, Thread_PWM);
   }


   // GPIO input thread for Fire Sensor detection
   status = ThreadCreateEx("Fire sensor", 4096, 100, (Address)Thread_Fire_GPIO_IN, NULL, &tid2);
    if(status != SUCCESS)
    {
       printf("%s>%s thread creation failed₩n", __FUNCTION__, Thread_Fire_GPIO_IN);
    }
    else
    {
       printf("%s>%s thread creation succeed₩n", __FUNCTION__, Thread_Fire_GPIO_IN);
    }

    // Relay Output thread for power supply control
    status = ThreadCreateEx("PWM", 4096, 100, (Address)Thread_Relay, NULL, &tid);
      if(status != SUCCESS)
      {
        printf("%s>%s thread creation failed₩n", __FUNCTION__, Thread_Relay);
      }
      else
      {
        printf("%s>%s thread creation succeed₩n", __FUNCTION__, Thread_Relay);
      }

    ThreadStart(tid, NO_WAIT);
    ThreadStart(tid2, NO_WAIT);
    ThreadStart(tid3, NO_WAIT);
    //UserRAMdiskInit( );

    //hal_main( );
}
```

1. 시스템의 시작 알림을 텍스트 출력을 통해 표현한다.
2. Thread는 총 3가지를 사용하므로, 각각의 Thread 변수들을 선언한다.
3. 정수형의 status를 선언하여 음파증폭기에 대한 PWM을 만들어내는 Thread를 만든다.

4. status의 상태에 따른 상황을 출력한다.
5. 화재감지 Thread인 Fire sensor를 선언한다.
6. status의 상태에 따른 상황을 출력한다.
7. Relay에 대한 PWM을 만들어내는 Thread를 선언한다.
8. status의 상태에 따른 상황을 출력한다.
9. 3개의 Thread를 시작한다.
10. Multi-Threaded Process로 진행된다.

### 3.3. 주요 함수의 흐름도




본 시스템의 주요 함수 흐름도는 위 그림으로 설명한다.

### 3.4. 기술적 차별성

- 화재 진압 후 처리가 용이
- 가스 형태의 소화기와 달리 밀폐된 공간에서 질식 방지
- 화재 진압에 필요한 물질 충전이 필요 없으므로 유지보수 비용이 적음
- 소화기에 대한 사용방법이 간단하고 화재 발생 시의 긴박한 상황에서도 사용하기 편리
- 레스토랑이나 음식점 주방 특성상 작업 공간이 좁은 곳에서 설치 시 다른 재료나 음식물 또는 공간을 오염시키지 않는 위생적 이점
- 잔여물이 발생하지 않고, 가스에 중독 될 위험이 없는 환경적으로 매우 친화적이고, 인체 안전성이 높음
- 유사 대체재나 경쟁 상품이 없음

## 4. 버전관리 이력

본 시스템의 Source code는 github에 공개하며, 주소는 다음과 같다.

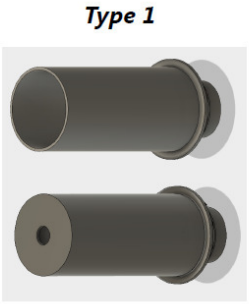
https://github.com/Choiyongare/RTOS_IoT_MSDE

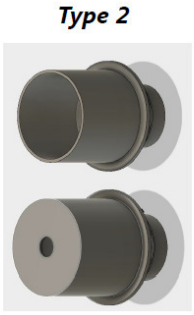
## 5. 개발 중 장애요인과 해결방안

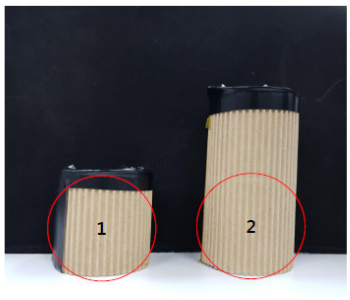
본 시스템의 목표를 달성하는 데 발생하는 장애요인은 다음과 같다.

- 화재 초기 진압을 위한 적절한 공명주파수 찾기
- 실시간적인 작동 반응성
- 제품 경량화 및 모듈화

첫째, 화재 초기 진압을 위한 적절한 공명주파수를 찾는 과정은 실험을 통해 해결하였다. 공명주파수를 만들어내기 위해, HW구성에 대한 프로토타입을 제작하여 비교분석을 실행하였다.

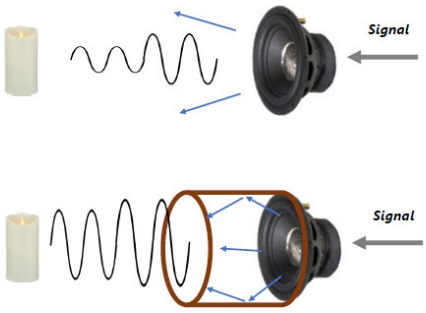


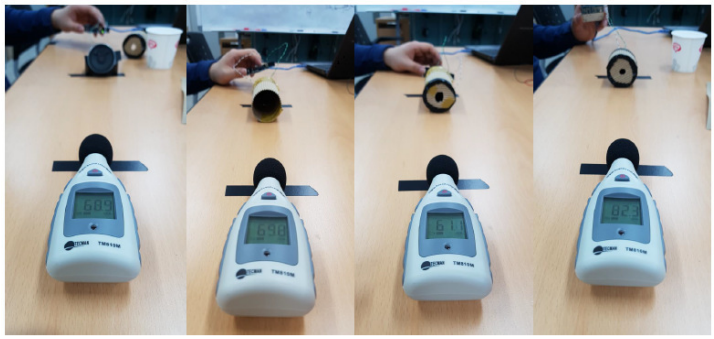




| Shape | Length | Hole | dB |
|---|---|---|---|
| X | X | X | 68.9 |
| Cylinder | 5cm | X | 81.4 |
| Cylinder | 5cm | O | 82.3 |
| Cylinder | 10cm | X | 69.8 |
| Cylinder | 10cm | O | 61.1 |

위 그림과 같이 1번은 실린더 길이를 5cm, 2번 실린더의 길이는 10cm로 두어 dB비교를 한 결과, 5cm가 최대 dB을 만들어 5cm를 선택하였다.



그 결과, 콜리메이터를 통해 스피커에서 나오는 음파를 한 곳으로 집중시켜 더 큰 음파를 만들어내도록 구성하는데 성공하였다. 다음으로, 본 시스템의 유효 소화 거리를 측정하기 위해 실제 화재 상황을 가장한 실험을 수행하였다.

| Hertz | Distance |
|---|---|
| 30Hz | X |
| 35Hz | 69cm |
| 40Hz | 80.5cm |
| 45Hz | 63.5cm |

결과적으로, 35Hz ~ 45Hz의 주파수에서 음파소화기 시스템이 안정적으로 역할을 수행하였다. 이를 통해, 화재 초기 진압을 위한 주파수 및 음파소화기 시스템의 HW목표를 달성할 수 있었다.

둘째, 실시간적인 작동 반응성을 올리기 위해 한컴MDS사의 NEOS(RTOS)를 사용하여 해결하였다. 본 시스템의 목표는 화재 초기 진압인 만큼 빠른 화재 감지 및 대처가 필요하다. 이에 RTOS를 사용하여, Multi-Threaded Process를 통한 실시간성을 가지게 되었다.

셋째, 제품 경량화 및 모듈화를 위해 제품 중량의 대부분을 차지하는 스피커의 무게를 0.8kg으로 선택하였고, 하나의 모듈로써 실제 가전제품에도 용이하게 탈착되도록 회로를 구성하였다.

**6. 개발결과물의 차별성**

- 실시간IoT에 적합한 시스템
- 화재안전 대책 방안에 대한 수요 만족
- 화재 발생이 가능한 산업에 쉽게 도입 가능
- 상대적으로 저전력 구동이 가능한 Realtime 회로 구성

**7. 단계별 개발계획 및 실제 참여인원 및 업무 분장**

| 업무 / 이름 | HW개발 | HW실험 및 비교분석 | 화재 연구 | NEOS 개발환경 | 알고리즘 및 함수 작성 | 제품 구입 및 시장조사 | 기록 및 보고서 작성 |
|---|---|---|---|---|---|---|---|
| 한승한 (팀장) | ■ | ■ | | ■ | ■ | | ■ |
| 박형주 | ■ | ■ | ■ | | | | |
| 최용래 | ■ | ■ | | | | ■ | ■ |
| 황형준 | | | ■ | ■ | ■ | | |
| 양재필 | | | | ■ | ■ | ■ | |