마이크로프로세서응용

기말 프로젝트 보고서

조 번호: 9조		
전산전자공학부	21900780	하정원
전산전자공학부	22100768	한나린

과목명	마이크로프로세서 응용
담당교수명	이강
제출일	2024. 6. 20.

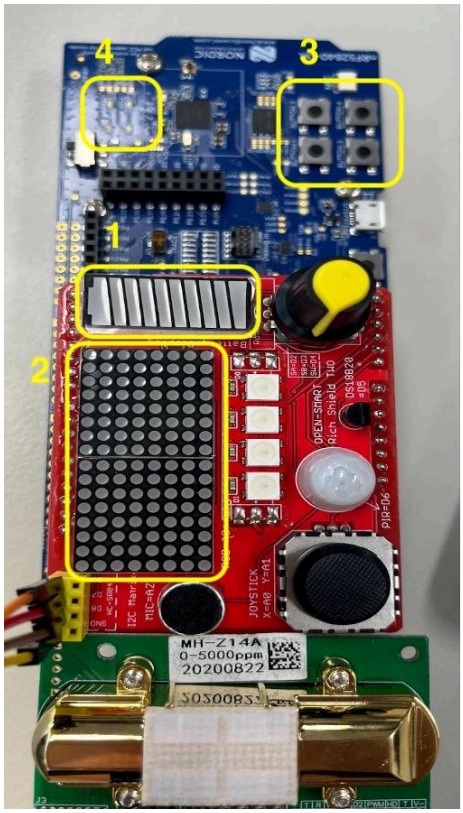
목차

1. 프로젝트 개요

1.1. 문제 정의

아파트는 한국에서 가장 널리 사용되는 주거 양식 중 하나로, 거주자들은 외부 방문자와의 소통을 위해 주로 인터폰 시스템을 사용한다. 이러한 시스템은 방문자가 아파트 공동 현관을 출입할 때, 거주자와 직접 소통할 수 있게 해주며, 거주자의 안전과 편의를 동시에 제공한다. 이번 마이크로프로세서 응용 기말 프로젝트에서는, 보편적 임베디드 시스템인 인터폰 시스템에 착안해, 주어진 움직임 감지 센서(PIR), 사운드 센서, 이산화탄소 측정 센서 및 기타 부가 장치들을 조합하여 인터폰 기능을 모방한 Valid Voice Detector(VVD)를 설계 및 구현하고자 한다. 음성(소리) 인식에 움직임 감지 및 이산화탄소 농도 측정을 통해 실제 사람이 접근하는지 여부를 판단해 보안 및 기능 향상을 도모하는 것이 이번 프로젝트의 주된 목표다. 이를 구현하기 위해 다양한 센서 및 장치들을 활용하고, 임베디드 시스템의 설계 및 프로그래밍 기술을 종합적으로 적용함으로써, 한 학기 동안 배운 내용을 실습하고 실제로 응용할 수 있는 기회를 마련하고자 한다.

1.2. 사용자 인터페이스 정의



사용자가 Valid Voice Detector(이하 VDD)를 이용하기 위해서는 보드 앞에서 움직임(Motion)을 보인다. 사용자의 움직임이 인식되면 (1)배터리 디스플레이에 색깔이 표시되어 VVD를 사용할 수 있는 상태라는 것을 알 수 있다.

사용자는 (2)LED Matrix에 표시되는 값을 통해 기기가 인식하는 음성에 관한 정보를 시각적으로 확인할 수 있으며, (3)버튼으로 정보 확인 모드를 선택할 수 있다. 모드는 총 3가지로 음성 주파수(Button 1), 숫자로 표시되는 볼륨(Button 2), 음성 인식 여부(Button 3)가 있다. Button 4를 누르게 되면 모든 기능이 리셋 되어 프로그램을 새로 시작할 수 있다.

버튼이 눌릴 경우, 버튼 넘버에 상응하는 넘버의 (4)LED가 약 3초간 깜박거려 사용자가 어떤 버튼을 눌렀는지 확인할 수 있다.

1.3. 사용된 입출력 장치 목록

- Battery Display
- Sound Sensor
- 8 * 16 LED Matrix
- Co2 Sensor
- PIR
- Button on nRF5240
- LED on nRF5240

2. 설계 및 구현

2.1. 개발환경 설명

2.1.1. 하드웨어 환경(마이크로프로세서 및 보드)

- Nordic Semiconductor 의 nRF52840DK 보드
- ARM Cortex-M4 32-bit processor with FPU, 64 MHz

2.1.2. 탑재된 시스템 소프트웨어 (RTOS 및 주요 라이브러리)

- RTOS: Zephyr
- 주요 라이브러리
  - 〈zephyr/kernel.h〉
  - 〈zephyr/device.h〉
  - 〈zephyr/devicetree.h〉
  - 〈zephyr/drivers/gpio.h〉
  - 〈zephyr/drivers/adc.h〉
  - 〈zephyr/drivers/uart.h〉
  - 〈zephyr/drivers/led.h〉
  - 〈zephyr/drivers/kscan.h〉

2.1.3. 개발에 사용된 언어 및 IDE

- IDE: VS code, nRF Connect
- 언어: C, ARM Assembly

2.1.4. 사용된 주변 장치 상세 내역

- Open Smart 의 Rich Shield Two 보드
  - Sound Sensor
  - PIR Sensor
  - 8 * 16 LED Matrix

    - Battery Display
- Co2 Sensor: Intelligent Infrared Carbon Dioxide Module (MH-Z14A)
- Button on nRF52840DK
- LED on nRF52840DK

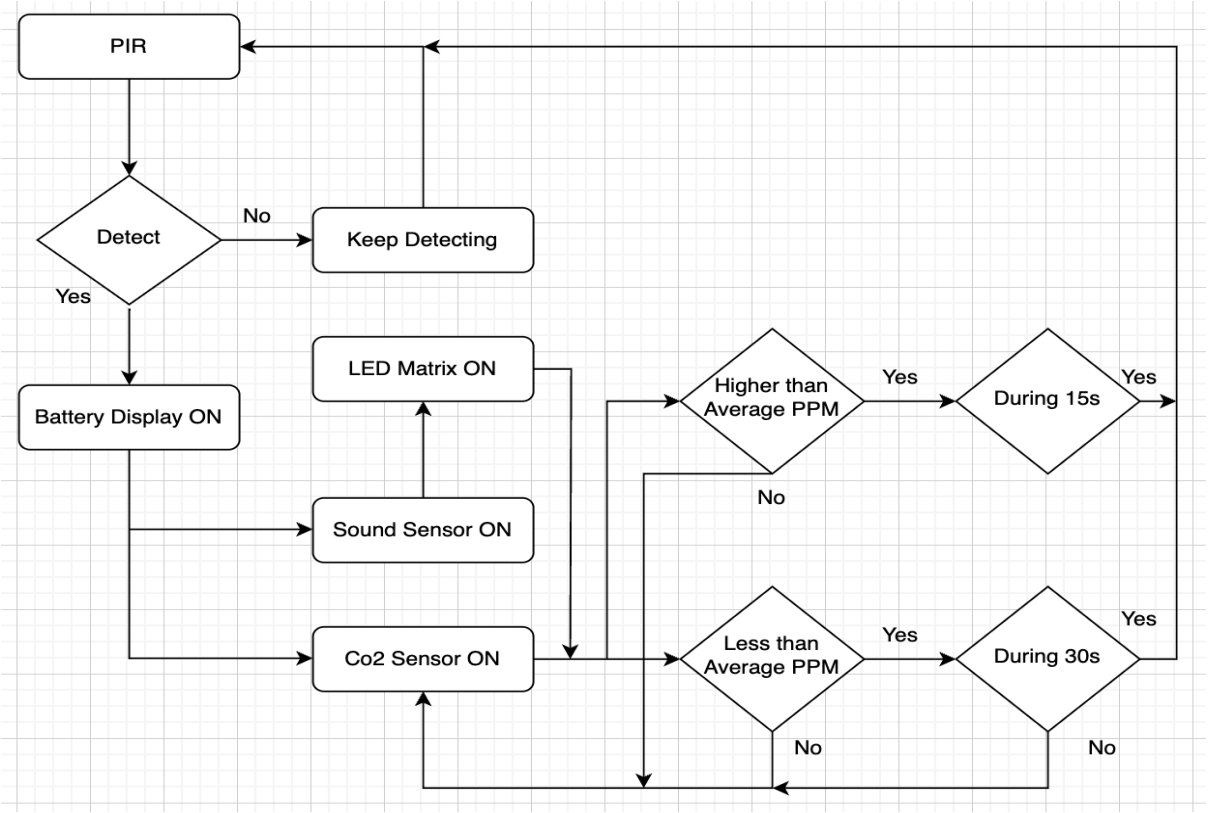
2.1.5. CMakeList.txt

```
cmake_minimum_required(VERSION 3.20.0)
find_package(Zephyr REQUIRED HINTS $ENV{ZEPHYR_BASE})
project(Main)

FILE(GLOB app_sources src/*.c)
target_sources(app PRIVATE ${app_sources})
```

2.2. 프로그램의 구조

2.2.1. 흐름도



2.2.2. 모듈 설정(Kconfig)

2.2.2.1. prj.conf

```
CONFIG_LOG=y
CONFIG_PRINTK=y

CONFIG_GPIO=y

CONFIG_ADC=y
CONFIG_ADC_ASYNC=y
CONFIG_ADC_NRFX_SAADC=y

CONFIG_I2C=y
CONFIG_LED=y
CONFIG_KSCAN=y
CONFIG_KSCAN_INIT_PRIORITY=95
CONFIG_HT16K33_KEYSCAN=y

CONFIG_UART_INTERRUPT_DRIVEN=y
CONFIG_UART_ASYNC_API=y
```

2.3. 주변장치

2.3.1. 사용된 메모리맵 (주변 장치별 접근 주소)

Sound Sensor	GPIO Port0.28 PIN (0x50000770)
PIR Sensor	GPIO Port1.7 PIN (0x50000A1C)
8 * 16 LED Matrix	I2C Address: 0x70
Battery Display	CLK: GPIO Port1.12 PIN (0x50000A30) DIO: GPIO Port1.13 PIN (0x50000A34)
Co2 Sensor	GPIO Port1.10 PIN (0x50000A28) GPIO Port1.08 PIN (0x50000A20)
Button on nRF52840DK	Button1: 0x5000072C Button2: 0x50000730 Button3: 0x50000760 Button4: 0x50000764
LED on nRF52840DK	LED1: 0x50000734 LED2: 0x50000738 LED3: 0x5000073C LED4: 0x50000740

2.3.2. 디바이스 트리(DeviceTree Overlay)

```
/ {
	aliases {
		pir = &pir0;
		gpio-clk = &gpioclk;
		gpio-dio = &gpiodio;
		myserial = &uart1;
	};

	gpiocustom {
		status = "okay";
		compatible = "gpio-keys";

		gpioclk: gpioclk {
			gpios = <&gpio1 12 GPIO_ACTIVE_HIGH>;
			label = "gpioclk P1.12";
		};

		gpiodio: gpiodio {
			gpios = <&gpio1 13 GPIO_ACTIVE_HIGH>;
			label = "gpiodio P1.13";
		};
	};
```

```
		pir {
			compatible = "gpio-keys";
			pir0: pir0 {
			gpios = <&gpio1 7 (GPIO_ACTIVE_LOW | GPIO_PULL_UP)>;
			label = "PIR Sensor";
			};
		};

		zephyr,user {
			io-channels = <&adc 4>;
		};
};

&adc {
		#address-cells = <1>;
		#size-cells = <0>;

		channel@4 {
			reg = <4>;
			zephyr,gain = "ADC_GAIN_1";
			zephyr,reference = "ADC_REF_INTERNAL";
			zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
			zephyr,input-positive = <NRF_SAADC_AIN4>; /* P0.28 */
			zephyr,resolution = <10>;
		};
};

&i2c0 {
		clock-frequency = <I2C_BITRATE_STANDARD>;

		ht16k33@70 {
			compatible = "holtek,ht16k33";
			reg = <0x70>;
			keyscan {
				compatible = "holtek,ht16k33-keyscan";
			};
		};
};

arduino_serial: &uart1 {
   status = "okay";
   compatible = "nordic,nrf-uart";
		current-speed = <9600>;
		pinctrl-0 = <&uart1_default>;
		pinctrl-1 = <&uart1_sleep>;
		pinctrl-names = "default", "sleep";
};

&pinctrl {
		uart1_default: uart1_default {
			group1 {
				psels = <NRF_PSEL(UART_RX, 1, 10)>;
				bias-pull-up;
			};
			group2 {
				psels = <NRF_PSEL(UART_TX, 1, 8)>;
			};
```

```
        };

        uart1_sleep: uart1_sleep {
                group1 {
                        psels = <NRF_PSEL(UART_RX, 1, 10)>,
                                <NRF_PSEL(UART_TX, 1, 8)>;
                        low-power-enable;
                };
        };
};
```

2.3.3. 주변 장치별 프로세서와 통신 표준 설명

Sound Sensor	GPIO, ADC
PIR Sensor	GPIO
8 * 16 LED Matrix	HT16K33 LED Driver, I2C
Battery Display	GPIO
Co2 Sensor	UART
Button on nRF52840DK	GPIO
LED on nRF52840DK	GPIO

2.4. 프로세스 및 인터럽트 관리

2.4.1. 인터럽트 설정 (우선순위, ISR 등)

Soft Reset Interrupt	AIRSR을 통해 SYSRESETREQ를 설정하여 Reset_Handler를 호출하는 Interrupt를 발생시킴
Button GPIO Interrupt	- GPIO Interrupt를 설정해 지정한 Callback 함수로 ISR이 실행되도록 함 - ISR: Button 넘버에 상응하는 LED를 깜박이게 하고, btn_flag의 값을 변경함
PIR GPIO Interrupt	- GPIO Interrupt를 설정해 지정한 Callback 함수로 ISR이 실행되도록 함 - ISR: PIR이 움직임을 감지해 값이 들어올 경우, motion_detected 값을 true로 변경함

2.4.2. Multi-Thread 사용 시 Thread별 역할과 통신

1) Sound Sensor Thread

본 프로젝트의 IoT는 Sound Sensor와 Co2 Sensor가 동시에 센싱을 진행하며 값을 읽어내야한다. 특히, Sound Sensor는 사용자가 목소리를 내어 말할 때 해당 음성을 감지하고, 주파수나 소리 세기를 LED Matrix에 출력해야하기 때문에 기기 사용 중일 때는 사용자의 목소리를 항상 읽을 수 있어야 한다. 따라서 Sound Sensor는 Thread를 통해서 구현되었다.

2) Co2 Sensor Thread

본 프로젝트의 IoT는 Sound Sensor와 Co2 Sensor가 동시에 센싱 진행하며 값을 읽어내야한다. 특히, Co2 Sensor는 사람의 음성을 포함한 주변의 모든 소리를 센싱하는 Sound Sensor가 인식한 소리가 사람의 목소리라는 것을 식별해낼 수 있는 방안이다. 따라서 Co2 Sensor는 Sound Sensor와 Multi-Thread로 동시에 동작하게 하게 함으로써 사용자가 말하지 않는 경우에는 일정 시간 이후 다시 PIR 감지 상태로 되돌아 갈 수 있게끔 기능 지원을 한다.

3) Multi-Thread 간의 통신

위의 두 Thread는 독립적으로 작동하다가 통신을 통해 PIR 센싱으로 돌아간다. Co2 Sensor를 통해서 사용자가 더 이상 말하지 않는다는 것이 식별이 되면, Co2 Sensor는 현재 PPM(Co2 센싱값)을 -1로 만든다. Sound Sensor는 주기적으로 이 PPM 변수를 읽으며 종료되는 시점을 감지한다.

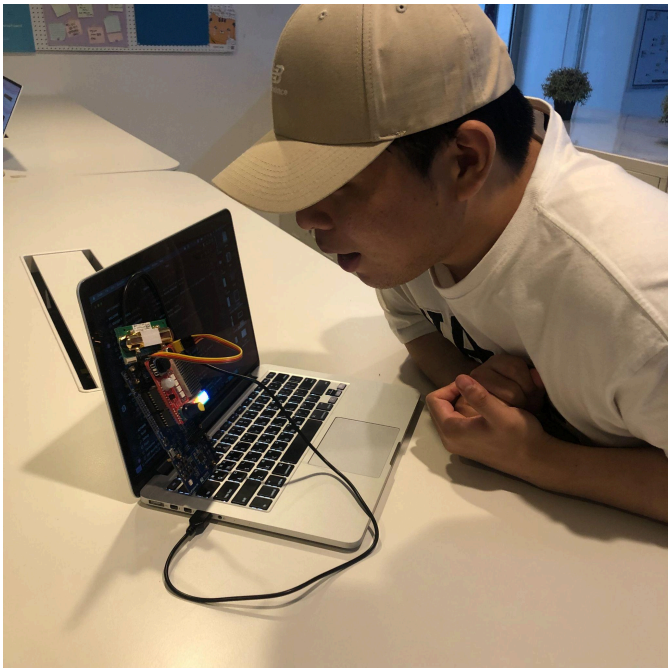
3. 구현 결과

3.1. 사용방법 설명

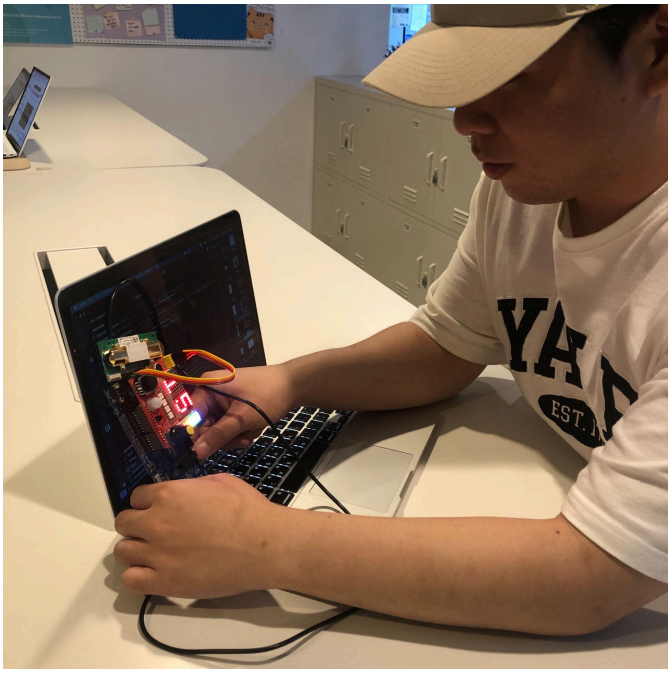
1. 사용자가 설치된 보드 앞으로 다가가면 PIR이 움직임을 감지한다. PIR이 움직임을 인식하지 못할 경우, PIR 근처에서 손을 휘젓는 등 움직임을 보인다.
2. Sound Sensor와 Co2 Sensor 방향으로 얼굴을 대고 말한다. Sound Sensor를 통해 음성을 인식시키고, Co2 센서를 통해 내뿜는 이산화탄소를 인식시키기 위함이다.
3. 만약, LED Matrix의 출력 방식을 바꾸고 싶다면 Button 1, 2, 3을 눌러 모드를 설정한다. 전체 기능을 리셋하고 싶다면 Button 4를 누른다.

3.2. 동작 시연 (이미지와 더불어 설명)

1. PIR이 움직임을 감지했으므로 센서가 소리와 이산화탄소를 센싱하도록 다가가서 말을 시작함



2. 버튼을 눌러 모드를 변경해 센싱한 소리의 값이 숫자로 표현되도록 함



3. 버튼을 눌러 모드를 변경해 센싱한 소리의 값이 주파수로 표현되도록 함

3.3. 자체 평가

3.3.1. 난이도 평가 (평가 기준에 근거하여 제시)

- *난이도*: 중상
- *근거 1*: PIR을 연결해 사용하는 오픈소스를 찾을 수 없었으며, 특히 nRF52840 보드에 PIR을 연결해 사용하는 예제 코드는 존재하지 않았다. 그렇기 때문에 Data Sheet와 PIN configuration을 참고해 Overlay를 작성하였다.
- *근거 2*: 주어진 Co2 Sensor는 센서가 설치된 환경의 이산화탄소 농도가 즉각적으로 센싱되지 않았다. 이러한 지연 시간(latency)은 IoT라는 사용자와 기기 간의 자연스러운 소통이 중요한 환경에서 큰 영향을 미친다. 본 프로젝트에서는 소음과 사람의 목소리를 구별할 수 있는 유일한 수단이 이산화탄소라고 판단했기에, 초기 프로젝트 계획에서 제시했던 Sound Sensor의 센싱 값과 Co2 Sensor의 센싱 값을 “&&” 로 검사해 사람의 목소리라고 판단하려던 부분을 수정했다. Co2 Sensor의 센싱 지연 시간을 측정하고, 이를 고려해 타이머(초)와 Sound Sensor의 센싱 값을 조합해 소음인지, 사람의 목소리인지, 사람의 목소리인 경우 말이 지속되거나 끝났는지를 판단했다.
- *근거 3*: 임베디드 프로그래밍은 기존의 소프트웨어 프로그래밍 방식과 다른 결을 보인다. 그간 해왔던 프로그래밍은 리눅스와 같은 편의성이 높고, 규모가 큰 운영체제가 제공하는 시스템 함수와 라이브러리를 기반으로 이루어졌다. 이번 프로젝트에서는 Zephyr 라는 소형 실시간 운영체제(RTOS)를 사용했기 때문에 새로운 시스템 함수를 공부해야만했다. 본 프로젝트에서는 시스템 함수 중 Thread 부분을 적극적으로 사용했고, 이와 관련한 공부는 Zephyr API 사이트에서 진행하였다.
- 근거 4: 하드웨어에 진행되는 임베디드 프로그래밍의 특성 상, nRF connect SDK와 같은 개발 툴을 새롭게 사용했다. 보드(또는 MCU)에 맞는 Configuration 과

부가적으로 사용한 주변 장치의 PIN Configuration 등도 적절히 설정해야 했다. 통신 방법 또는 주변 장치의 하드웨어적 특성에 따른 지연 시간도 측정하며, 소프트웨어 상에서 이를 어떻게 핸들링할 것인지를 다뤄야 했다.

3.3.2. 계획 대비 완성도

기존 계획 대비 완성도는 95%로, 부족한 5%는 Co2가 실시간 값을 얻지 못한다는 점 때문에 삭감되었다.

3.3.3. 조원의 개인 기여 내용

조원	기여 내용	상호평가
하정원	1. 프로젝트 아이디어 제공 및 설계 2. PIR Sensor, Sound Sensor + LED Matrix 구현 3. 전체적인 구현 방향 설정 및 리딩 4. 코드 병합 및 문서화	1.0
한나린	1. 프로젝트 아이디어 제공 및 설계 2. Co2 Sensor, Battery Display, Button Interrupt 구현 3. 코드 리팩토링 및 문서화	1.0

4. 부록

4.1. 작성된 파일 목록 및 설명 (프로젝트 README 파일 내용)

4.1.1. 소스코드 리스트

```
src
├ batterydisplay.c
├ batterydisplay.h
├ button.c
├ button.h
├ co2.c
├ co2.h
├ led.c
├ led.h
├ main.c
├ pir.h
└ value.h
```

4.1.2. 빌드 방법

1. nRF Connect로 해당 프로젝트를 Open한다.
2. Add Build Configuration를 눌러 Build환경을 설정한다.
    a. Board: nrf52840dk_nrf52840

b. configuration: prj.conf
c. DeviceTree Overlay: nrf52840_nrf52840.overlay

3. Press Build Configuration

4.1.3. 소스 작성자 정보 및 날짜

- 하정원: 한동대학교 전산전자공학부 hajeongwon77@gmail.com
- 한나린: 한동대학교 전산전자공학부 lynnie21@handong.ac.kr

4.1.4. 사용한 오픈소스 정보

https://github.com/UmileVX/IoT-Development-with-Nordic-Zephyr

Sensor에 관한 대부분의 코드는 위 깃허브의 예제 코드를 참고해 수정함

4.1.5. Copyright 정보가 포함된 텍스트 파일

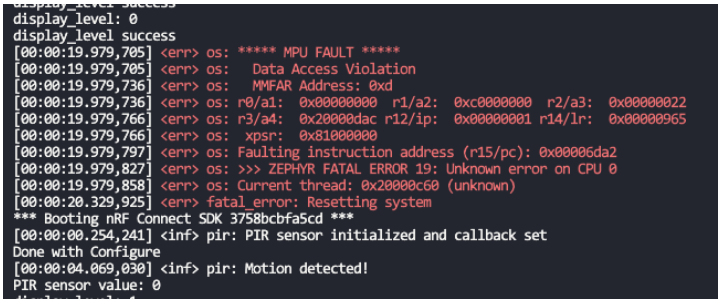
GNU Public License Version3, LICENSE 파일로 작성

4.1.6. 프로젝트 Workspace 루트

```
.
├── CMakeLists.txt
├── LICENSE
├── nrf52840_nrf52840.overlay
├── prj.conf
├── sample.yaml
└── src
    ├── batterydisplay.c
    ├── batterydisplay.h
    ├── button.c
    ├── button.h
    ├── co2.c
    ├── co2.h
    ├── led.c
    ├── led.h
    ├── main.c
    ├── pir.h
    └── value.h
```

4.2. 개발과정에서 만났던 에러가 있는 경우, 에러 메시지와 그 대처 방안 설명

4.2.1. MPU(Memory Protection Unit) FAULT



1) Assembly 코드에서 Register 값의 변경이 이뤄져 Memory Fault가 발생함
   - 대처방안: Register 값을 따로 저장해 Branch 한 곳에서 Register 값을 변경하더라도 원래의 주소로 돌아올 수 있도록 함
2) Thread Creation과 Thread Start에서 Thread ID와 관련해 Memory Fault가 발생함
   - Thread를 만들면 할당되는 Thread ID를 변수에 제대로 저장하고, Start를 실행할 때 이 값을 참조할 수 있도록 함

4.3. 개인 소감

- **하정원**: 임베디드 프로그래밍이란 이런 것인가?라는 것을 느꼈다. 지금까지는 좋은 컴퓨터와 좋은 OS에서 여러 서비스들을 제공 받으며 편하게 프로그래밍 했었는데, 임베디드 프로그래밍은 함수 하나, 메모리 하나가 프로그램을 실패 시킬 수도 있고 속도를 현저히 저하시킬 수도 있다는 것을 많이 느꼈다. 특히 인상적이었던 부분은, Zephyr RTOS가 제공하는 시스템 함수를 사용했을 때였다. 지금까지는 아무 거리낌 없이 thread를 위해서는 pthread, print를 위해서는 printf 그리고 프로세스를 위해서는 fork(), 이런 식으로 시스템 함수를 사용해왔는데, 이번에도 당연하게 이런 식으로 생각했다가 OS가 다르기에 시스템 함수도 다르다는 당연한 이론을 몸소 체험하였다. 이러한 관점에서 컴퓨터 공학생들은 임베디드 수업을 들어야하구나라는 것을 깨달았다.
- **한나린**: 기말 프로젝트를 설계할 때, 수업 시간에 배운 내용을 십분 활용하기 위해 가장 보편적이고 흔하게 사용하는 아파트 공동현관의 인터폰이 떠올랐다. 카메라와 키패드가 없을 뿐, 음성 인식이라는 큰 틀에서 주어진 보드와 주변 장치들을 활용해 충분히 인터폰을 모방한 프로그램을 구현해볼 수 있을 것 같았기 때문이다. 막상 구현을 시작하자, 사용하는 주변 장치 하나하나를 Data sheet를 참고해 이해하고, 보드에 맞는 overlay를 작성해야 했는데, 이 작업이 꽤나 복잡했다. 특히나, PIR이 움직임을 감지하면 RGB LED를 깜박여 움직임을 감지했음을 시각적으로 보여주기 위한 작업을 성공시키고 싶었지만, 예제 코드나 여러 사이트를 참고했음에도 결국 RGB LED 구동에 실패해 아직도 아쉬움이 남는다. 전반적으로 하드웨어와 직접 연결된 소프트웨어 제어를 경험해볼 수 있어 새로웠다. 임베디드 시스템 프로그래밍은 data sheet 참조와 여러 개발 커뮤니티에서 오고가는 질문과 답변들을 잘 참고해 적용해 봐야 한다는 게 가장 큰 특징인

것 같다. 소규모 + Real Time OS인 Zephyr의 특성에 맞춘 라이브러리나 API 사용, Conf와 Overlay 설정을 도와주는 nRF Connect SDK를 사용하는 것도 처음엔 복잡하게 느껴졌지만, 사용을 계속할수록 적응할 수 있었다. 여러모로 Low level 에서의 프로그래밍을 통해 컴공의 정수를 맛보는 한편, High level 에서의 프로그래밍에 대한 감사함을 느낄 수 있었다.

4.4. 개발 과정에서 참조 했던 리소스

4.4.1. 블로그 등 URL, 하드웨어 업체 제공 공식 문헌

1) zephyrproject-rtos. (2016). Zephyr Kernel.h. GitHub. Retrieved from: https://github.com/zephyrproject-rtos/zephyr/blob/main/include/zephyr/kernel.h
2) Zephyr System Function & APIs. Retrieved from: https://docs.zephyrproject.org/apidoc/latest/group__kernel__apis.html
3) UmileVX. (2024). IoT-Development-with-Nordic-Zephyr. GitHub. Retrieved from: https://github.com/UmileVX/IoT-Development-with-Nordic-Zephyr/tree/main

4.4.2. Youtube, StackOverflow, 개발 Community, Chat-GPT 등의 활용 내역

1) Nordic DevZone Q&A. Controlling WS2812 LEDs with NCS on nRF52832DK with gpios (not SPI). Retrieved from: https://devzone.nordicsemi.com/f/nordic-q-a/88603/controlling-ws2812-leds-with-ncs-on-nrf52832dk-with-gpios-not-spi